Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Preparação da Dissertação de Mestrado em Engenharia Informática

2º Semestre, 2007/2008

Análise de imagens tomográficas: visualização e paralelização de processamento

Tiago João Gonçalves Cadavez

Nº25963

Orientador

Prof. Doutor Pedro Abílio Duarte de Medeiros

31 de Julho de 2008

Nº do aluno: 25963

Nome: Tiago João Gonçalves Cadavez

Título da dissertação:

Análise de imagens tomográficas: visualização e paralelização de processamento

Nº do aluno: 25963

Nome:

Tiago João Gonçalves Cadavez

Título da dissertação:

Análise de imagens tomográficas: visualização e paralelização de processamento

Palavras-Chave:

- Paralelização de aplicações
- OpenMP
- Tomografia
- Visualização
- OpenDX

Keywords:

- Application parallelization
- OpenMP
- Tomography
- Visualization
- OpenDX

Agradecimentos

Agradeço vivamente a todos os que me ajudaram a finalizar este projecto:

À Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, de forma particular a todos os professores que, desde a minha entrada na Faculdade ao longo do curso, me transmitiram a formação de base nesta área de estudo.

Ao Prof. Dr. Pedro Medeiros pela imensa ajuda para encontrar o caminho certo, pela disponibilidade total para clarificar duvidas e pela sua atitude persistente de encorajamento.

Ao Departamento de Engenharia de Materiais pela sua atitude de colaboração interdisciplinar, em particular, o Prof. Dr. Alexandre Velhinho e a Sónia Ferreira que, utilizando o programa proposto nesta tese, contribuíram que ela não fosse construída a partir de arquétipos teóricos, mas sim orientada para uma realidade experimentada na prática.

À Sun Microsystems e Sun Microsystems Portugal dentro da "Sun Worldwide Marketing Loaner Agreement #11497" por quem este trabalho foi parcialmente suportado.

Aos meus colegas mais próximos e família, com quem partilhei ideias e recebi encorajamento.

Resumo

A micro-tomografia de raios-X por radiação do sincrotrão é uma técnica bem desenvolvida no domínio da medicina, e mais recentemente, foi adoptada em outras áreas, nomeadamente na Engenharia de Materiais. É uma técnica não destrutiva, que permite analisar a estrutura interna de componentes. O objecto em estudo é alvo de um feixe de radiação por toda a superfície que penetra no material, e um conjunto de detectores vai registando a intensidade dos raios à medida que o objecto vai rodando. Este procedimento origina um ficheiro de dados, que pode ter uma dimensão da ordem dos gigabytes e que necessita de ser processado para visualização da estrutura interna. Atendendo ao grande volume de dados e à complexidade de alguns algoritmos de processamento, certos tipos de processamentos podem mesmo levar dias. O programa do cientista francês Gerard Vignoles, Tritom, processando os dados tomográficos sequencialmente, demorava muito tempo em algumas operações. Num esforço anterior, Paulo Quaresma optimizou e paralelizou algumas das operações mais demoradas, usando um agregado (clusters) de computadores e programação baseada em troca de mensagens. Nesta tese, paralelizaram-se as operações mais pesadas do Tritom utilizando o modelo de memória partilhada, concretizado através da ferramenta OpenMP. Esta aceleração de obtenção de resultados é vantajosa para a investigação sobre os materiais e permite tirar partido da introdução de multi-processadores nas arquitecturas de computadores pessoais comuns. Foram realizados testes para análise das melhorias de tempo de execução com este método. Nesta tese, também se integrou também o Tritom num ambiente de visualização de dados gráfico e interactivo de nome OpenDX que facilita muito a utilização do programa aos menos experientes. O utilizador pode escolher os processamentos que deseja realizar sob a forma de módulos e pela ordem que quiser tudo num ambiente gráfico. Permite também a visualização tridimensional de dados que se torna vital para percepcionar alguns fenómenos nos objectos de estudo. Foram também criados alguns novos módulos a pedido dos investigadores de Engenharia de Materiais. O Tritom, assim paralelizado, oferecerá aos cientistas da área de materiais uma boa ferramenta de análise de imagens tomográficas com interacção simples e intuitiva. Poderão, dispor de uma mais-valia para leituras rápidas dos seus objectos de estudo sem recorrerem a clusters ou configurações de computadores complexas e pouco acessíveis.

Abstract

The X-ray microtomography with radiation of the synchrotron is a well developed technique in the medicine area, and was recently adopted in other areas like Materials Engineering. It is a non destructive technique that allows the analysis of the objects internal structure. This technique consists of target the object in study with a radiation beam to its entire surface that penetrates the material and a set of detectors registers the beams intensity as the object rotates. This procedure originates a data file, sometimes with the size of gigabytes, which needs processing for the visualization of the materials internal structure. Due to the great volume data and the complexity of the processing steps, some of these computations can take days. In this thesis, the program Tritom that sequentially processes the tomographic data, is used. The execution of some of the steps of this program is very long. Paulo Quaresma optimized and parallelized the longest operations, using message passing approach in clusters environments. In this thesis, the Tritom's longest operations were parallelized using sharedmemory model by the library OpenMP, with very significant reduction in processing time. This acceleration in obtaining the results is very useful for the research in Materials Science and the choice of OpenMP allows one to take advantage of the introduction of multi-processors in the desktop personal computers. In this thesis, Tritom was also integrated in a graphical data visualization environment called OpenDX that eases the use of this application to the less experienced. The user may choose the operations he wishes to process and in which order using modules on a graphical environment with mouse and menus interaction. It also allows the three-dimensional data visualization which is vital to analyze certain phenomena in the object of study. By request of Materials Engineering researchers, some new data processing modules were developed. This Tritom version has proved to be a very useful tool for tomographic data analysis with a simple and easy interaction. Materials Engineering researchers will be able to process their data fast without having to use clusters or complex computer configurations.

Índice

Figuras

1. Introdução

O principal objectivo desta tese foi construir uma aplicação que permitisse a um especialista na área de Engenharia dos Materiais processar eficientemente dados obtidos através da técnica SXMT (Micro-tomografia de raios X com radiação do sincrotrão). O trabalho incidiu sobre duas áreas fundamentais:

- Diminuir do tempo de execução das fases de processamento mais demoradas. Para este efeito, efectuou-se a paralelização das partes do programa que efectuam esse processamento. Para esse efeito, a abordagem usada foi baseada em multi-processadores de memória partilhada.
- Permitir a um especialista da área de Materiais, uma utilização cómoda do programa, fornecendo-lhe uma interface gráfica em que se pode configurar o tratamento a efectuar, observar resultados intermédios tridimensionalmente e modificar parâmetros de tratamento.

1.1 Microtomografia de Raios-x

Como refere A.Velhinho em [2] a microtomografia de raios X com radiação e sincrotrão (SXMT) constitui um desenvolvimento relativamente recente da tomografia computorizada (CT). Esta última consiste numa técnica desenvolvida no âmbito da imagiologia em medicina onde se obtém, através do processamento de informação recolhida após o bombardeamento por raios X do objecto de estudo, uma imagem tridimensional. A aplicação na qual se baseia esta tese, tem como objectivo o processamento de imagens produzidas pela técnica SXMT e

como tal é feita uma breve descrição da microtomografia computorizada na qual se baseia, para melhor compreensão do contexto.

O princípio da tomografia computorizada consiste na medição da distribuição espacial de uma determinada grandeza física do objecto estudado, e obtenção, a partir desses dados, de imagens livres de sobreposições [3]. O objecto a ser estudado, na tomografia de materiais roda sobre si próprio e é alvo de radiação X, que pela sua natureza penetrante, é absorvida parcialmente em profundidade pelo material exposto, avaliando a sua estrutura pela intensidade da radiação após a sua penetração. A informação é gerada por camadas finitas, e cada camada é processada em separado através de um algoritmo, para mais tarde ser reconstruída como imagem tomográfica. Estas camadas bidimensionais podem ser combinadas para obtenção de uma imagem tridimensional.

A figura 1 apresenta simplificadamente componentes de um sistema tomográfico onde os feixes emitidos através de uma fonte são observáveis. Projectam-se numa amostra que se encontra em cima de uma plataforma giratória para a movimentação do objecto. No caso de imagens estudadas neste trabalho, estas amostras contêm reforços de outros materiais no seu interior, interessando aos cientistas de Materiais, informação sobre estes reforços. A imagem bidimensional é capturada e intensificada numa câmara, enviada para processamento, e reconstruída tridimensionalmente. Releva-se também a importância da distância entre a amostra e o emissor de radiação para obtenção de uma boa nitidez.

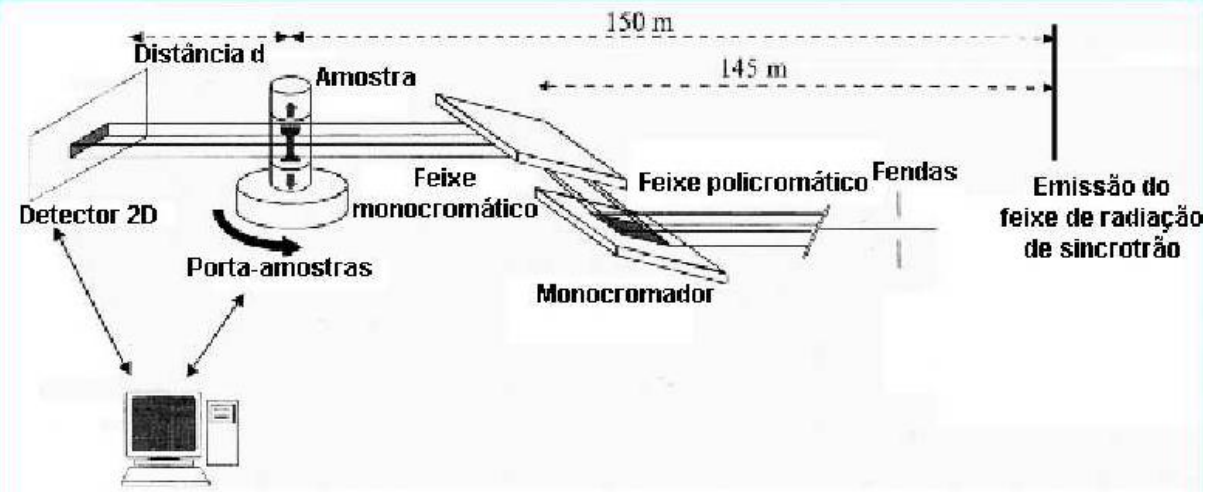

Figura 1: Representação esquemática de um sistema tomográfico. Retirado de [2].

Os dados são guardados em bruto pelo que, para que a informação possa ser visualizada mais tarde se impõe o uso de um programa que os processe. Atendendo ao grande volume de dados e complexidade dos algoritmos de processamento que podem demorar mesmo vários dias na computação, a paralelização é uma boa alternativa para distribuir a carga e acelerar o processo.

1.2 O programa tritom

Nesta secção descreve-se o programa de processamento de dados tomográfico (Tritom), primeiramente na versão sequencial e seguidamente aborda-se as fases para a sua paralelização.

1.2.1 Versão sequencial e sua paralelização

Esta tese assenta num programa desenvolvido por G. Vignoles [18] criado para processamento computorizado de dados tomográficos denominado Tritom e que posteriormente foi paralelizado numa tese de mestrado realizada por Paulo Quaresma [1]. A presente investigação continuou o trabalho realizado por Paulo

Quaresma recorrendo também à paralelização, mas com ênfase no uso de multi-processadores de memória partilhada.

O programa base Tritom de que partiu o trabalho referido, encontrava-se inicialmente, num único ficheiro em linguagem C, para a realização do processamento estritamente sequencial num fluxo de execução. Os dados eram lidos de um ficheiro em memória secundária para memória principal onde se dava o tratamento destes. Era dada a opção ao utilizador de guardar dados intermédios em memória secundária.

O programa disponibilizava um menu que fornecia as opções de processamento, onde se destacavam:

- Bi-segmentação da imagem (conversão de imagem em branco, preto, cinzento).
- Limpeza de imagem (eliminação de ruído).
- Histerese (dilatação do preto e branco no cinzento).
- Percolação de imagem (cor branca considerada como poros; eliminação de porosidade fechada).
- Estatísticas de Granulometria (determina a posição e dimensão dos reforços aplicados na amostra analisada).
- Recombinação de imagens e aplicação de máscaras.

Atendendo à disponbilidade deste programa e à crescente utilização da tomografia em diversas áreas ciêntificas, tornou-se relevante o desenvolvimento descrito nesta tese em que se procura diminuir o tempo de execução das operações mais longas bem como introduzir facilidades de visualização de dados. O trabalho realizado operacionalizou a funcionalidade até agora conseguida através de troca de mensagens, com memória partilhada num computador dispondo de uma arquitectura multi-processador. A eficácia deste projecto permitirá além da economia de tempo, a economia de meios, sem comprometer a qualidade.

Foi integrado no programa Tritom uma ferramenta que facilita a interacção entre o utilizador e a aplicação, para que esta se torne acessível a qualquer pessoa que não

esteja especialmente familiarizada com o programa. É ainda disponibiliza uma visualização de resultados sob forma tridimensional.

1.2.2 Paralelização

Relativamente à distribuição do trabalho entre vários processadores numa computação, duas das formas de paralelização mais comuns são a geométrica e a funcional, de seguida referidas com brevidade:

❖ **Paralelização Geométrica (ou baseado nos dados)**

A paralelização geométrica, escolhida para o desenvolvimento do trabalho nesta tese, constituiu a estratégia usada em [1] para paralelização no Tritom e baseia-se na divisão dos dados em blocos para processamento. Cada bloco é atribuído a um processador que executará as instruções necessárias sobre esse sub-bloco como demonstra o esquema da figura 2.

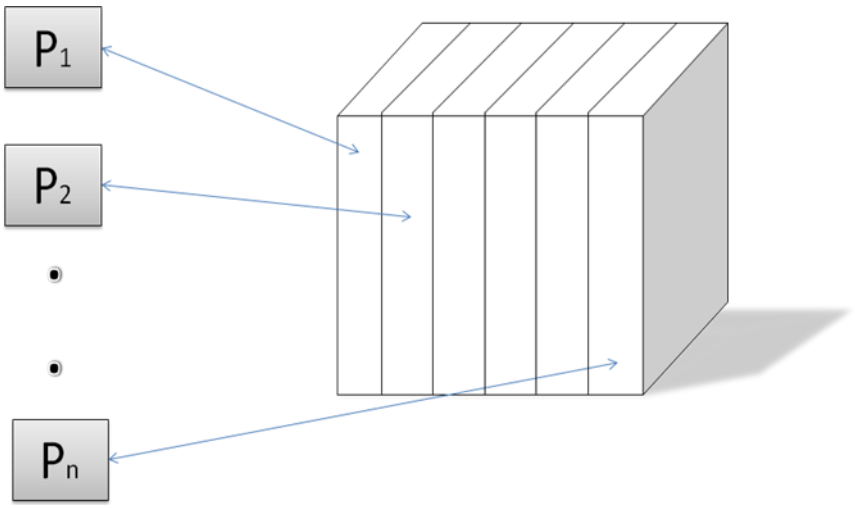

Figura 2: Esquema de paralelização geométrica

❖ **Paralelização Funcional (ou baseado no controlo)**

Esta forma de paralelizar centra-se nas instruções. Em vez de distribuir os dados por vários processadores como na paralelização geométrica, as instruções que se aplicam aos mesmos é que são distribuídas por cada processador. Este método não é tão simples de aplicar como o anterior e é muitas vezes difícil, organizar o programa de fora a dividir o trabalho em várias partes, permitindo a execução simultânea de operações diversas sobre diferentes partes de dados.

No presente trabalho, usou-se exclusivamente paralelização geométrica.

❖ **Comunicação entre processos**

Entendendo que a paralelização constitui uma boa alternativa para distribuir a carga e acelerar operações, impõe-se uma reflexão sobre métodos de comunicação de processos mais divulgados:

- ❖ Partilha de memória entre vários processadores.
- ❖ Troca de mensagens, normalmente entre máquinas distintas.

Acerca da primeira poderemos apresentar as seguintes caracteristicas:

- É preferida nas arquitecturas multi-processador
- Partilha com outros processadores a mesma memória física
- Sincroniza com outros processadores através de variáveis partilhadas de controlo.

A paralelização através de trocas de mensagens, opera de diferente modo:

- Realiza-se através de processos independentes com memória estritamente local
- Os processos comunicam através de envio e recepção de mensagens
- As operações de transferências de dados têm que ser coordenadas (um envio implica sempre a recepção correspondente).

1.2.3 Visualização

A natureza dos vários problemas a resolver na análise de imagens tomográficas, requer que o utilizador do sistema escolha quais as operações a aplicar aos dados, a ordem na qual devem ser executadas e os parâmetros para cada uma delas. Alguns parâmetros podem mesmo depender de resultados obtidos anteriormente.

Torna-se necessário o uso de uma ferramenta que permita de forma acessível a um utilizador não especialista na área da computação, construir sequências de operações sobre um volume de dados e definir facilmente os parâmetros dos algoritmos. Devido à grande variação de resultados conforme os parâmetros, a visualização dos dados em passos intermédios, ganha bastante relevância. Torna-se indispensável também que esta ferramenta possibilite a introdução de novos módulos de operações, em adição aos possíveis já existentes e que esses novos módulos possam encapsular processamento paralelo para acelerar a computação das operações demoradas.

1.3 Solução apresentada

Nesta secção resume-se o trabalho desenvolvido, começando-se por referir as estratégias de paralelização usadas e descrevendo as opções tomadas para a visualização de dados**.**

1.3.1 Paralelização

Como se afirmou anteriormente, o programa inicial para processamento de imagens tomográficas foi paralelizado na tese de Paulo Quaresma, recorrendo à técnica de passagem de mensagens através do MPI, com o objectivo de acelerar o seu processamento num *cluster* com vários computadores.

Nesta tese, como foi já referido, foi realizada uma paralelização usando partilha de memória recorrendo à ferramenta OpenMP. Este processo implicu o uso de apenas um computador para correr a aplicação, uma vez que a partilha de memória o obriga. Preferencialmente a aplicação deverá possuir uma arquitectura com multi-processador que lhe permitirá obter as vantagens do desejado paralelismo real.

Utilizando outras ferramentas, as operações paralelizadas foram as mesmas da tese anterior, nomeadamente limpeza de imagem e estatísticas de granulometria. Estas operações foram escolhidas por serem as que exibem um maior tempo de execução. O processamento da limpeza de imagem continuou a ser realizado em blocos independentes pois os dados não necessitam ser ligados. Na granulometria, porém, introduziram-se alterações e ultrapassaram-se limitações encontradas na versão anterior. A identificação dos elementos encontrados no material foi melhorada para que sejam reconstruídos reforços que pertencem a zonas de dados analisados por processos distintos.

1.3.2 Visualização

Nesta tese, o tritom paralelizado foi integrado numa ferramenta que permite visualização tridimensional de imagens tomográficas, fácil interacção gráfica com um utilizador inexperiente em computação e introdução de novos módulos permitindo computação paralela.

A escolha da ferramenta recaiu para o Open Data Explorer mais conhecido por OpenDX [36] que satisfaz todos os requisitos mencionados anteriormente. A vantagem desta ferramenta sobre outras com as mesmas funcionalidades é o facto de este software ser livre, de código aberto e ser vastamente usado na área científica.

A solução desenvolvida permitiu construir uma 1ª versão de um Problem Solving Environment (PSE) que simplifica a interacção do utilizador com a aplicação Tritom. Esta interacção passou a ser feita graficamente, viabilizando ao utilizador a fácil definição dos parâmetros da análise dos materiais, e a selecção de um fluxo de trabalhos onde poderá escolher as operações sobre os dados tomográficos a realizar, sendo possível a visualização de resultados intermédios.

1.4 Principais contribuições

De seguida descrevem-se as principais contribuições alcançadas nesta tese. Estas recaiem sobre a paralelização e consequente aceleração do processamento tomográfico e visualização de dados tomográficos num ambiente de fácil de usar para o utilizador

1.4.1 Paralelização

Com este trabalho, obteve-se uma optimização significativa do programa em referência para o processamento de imagens tomográficas. Como atrás foi dito, o processamento destas imagens, decorria em ritmo vagaroso podendo chegar a demorar dias, uma vez que se trata de dados de grande dimensão e serem usados algoritmos bastante pesados. A aceleração deste processamento permitirá um mais rápido avanço dos trabalhos realizados pelos Engenheiros de Materiais.

A paralelização do programa através da memória partilhada conduziu a melhorias significativas no âmbito da utilização deste programa, que como sabemos, em

algumas áreas científicas joga com materiais de efectiva aplicabilidade. Efectivamente por decorrer num ambiente de memória partilhada, poderá ser usado em computadores
de uso comum que recentemente começaram a ser dotados de uma arquitectura multi-processador, aproveitando-se assim completamente o total poder computacional da máquina.

Em síntese, poderão relevar-se os seguintes contributos relativos à paralelização:

- Maior rapidez e rigor em estudos inscritos na engenharia de materiais.
- Melhor acessibilidade de meios através do uso de apenas uma máquina de secretária para o processamento.
- No caso da granulometria, obtenção de resultados com melhor qualidade.

1.4.2 Visualização

Houve uma significativa melhoria na facilidade de utilização deste programa de processamento tomográfico paralelizado com a sua integração no OpenDX. Através deste ambiente, a interacção com o programa foi bastante facilitada para o utilizador, que intuitivamente poderá executar as operações que deseja e pela ordem que deseja num ambiente gráfico, para isso precisa apenas introduzir módulos no programa visual e interligá-los conforme o processamento que deseja. Pode visualizar os dados de forma tridimensional o que possibilita analisar certas características não visíveis em simples recortes bidimensionais da imagem. O programa permitiu visualizar resultados intermédios do processamento e interactivamente ajustar parâmetros fundamentais para a execução ideal das operações seguintes.

Em síntese, poderão relevar-se os seguintes contributos relativos à visualização:

- Melhoramento da facilidade de utilização da aplicação, através da integração de uma interface, permitindo interacção gráfica e intuitiva ao utilizador.
- Visualização de resultados em 3-D
- Visualização e ajuste de parâmetros de resultados intermédios.

1.5 Organização da tese

Esta tese está organizada da seguinte forma:

- Capítulo 1 introdução
- Capítulo 2, apresentação de um conjunto de trabalhos relativos quer na área de processamento paralelo, quer na dos sistemas de visualização de dados.
- O capítulo 3 apresenta uma visão geral do sistema construído.
- No capítulo 4 descreve-se a realização dos vários componentes do sistema e apresentam-se resultados que mostram a diminuição dos tempos de execução das fases mais demoradas. Também é discutida a parte da visualização, presentando-se como exemplo concreto de um estado na área de Engenharia de Materiais que utilizou o sistema construído.
- Finalmente o capítulo 5 apresenta as conclusões e as perspectivas de trabalho futuro.

2 Trabalho relacionado

Neste capítulo apresenta-se a envolvente deste trabalho, nomeadamente referenciando modelos de paralelização, arquitecturas paralelas, ferramentas para a paralelização, ferramentas para a visualização de dados científicos e trabalho relacionado com o processamento de dados tomográficos e respectiva visualização.

2.1 Arquitecturas Paralelas e desempenho

Um sistema paralelo caracteriza-se pelo objectivo de optimizar o desempenho de execução recorrendo a fluxos paralelos de processamento. Para melhor o caracterizar, poderemos recorrer à Taxonometria de Flynn pois é uma taxonomia de computadores paralelos que fornece uma forma de identificar importantes características de um sistema.

A taxonometria de Flynn [4] tornou-se muito popular na classificação para arquitecturas de computadores. De acordo com este esquema há dois tipos de informação que chegam ao processador: instruções e dados. O fluxo de instruções é definido como a sequência de instruções processadas. O fluxo de dados é definido como tráfego de dados trocados entre a memória e a unidade de processamento. De acordo com Flynn tanto um como o outro podem ser singulares ou múltiplos. A relação entre os fluxos de instruções e dados permite obter quatro classificações distintas:

- *Single Instruction Single Data* (SISD) – Uma instrução é executada por apenas um processador. Trata-se do caso de um computador com apenas um processador que executa uma instrução de cada vez com um único fluxo de dados.

- *Single Instruction Multiple Data* (SIMD) – A mesma instrução é executada por vários processadores através de uma estrutura de controlo sequencial que define os passos de processamento aplicados a vários fluxos de dados. Este modelo é usado em máquinas vectoriais e processadores matriciais (ex: sistemas de equações, analise de imagem, etc.)

- *Multiple Instruction Single Data* (MISD) – Esta categoria refere múltiplas instruções, mas apenas um fluxo de dados. Resultados produzidos por cada unidade de instruções são passados à unidade seguinte. Este modelo é raramente usado.

- *Multiple Instruction Multiple Data* (MIMD) – Cada processador pode executar diferentes instruções que normalmente operam sobre diferentes conjuntos de dados. Este modelo divide-se em sub-modelo de partilha de memória e troca de mensagens que serão aprofundados mais à frente.

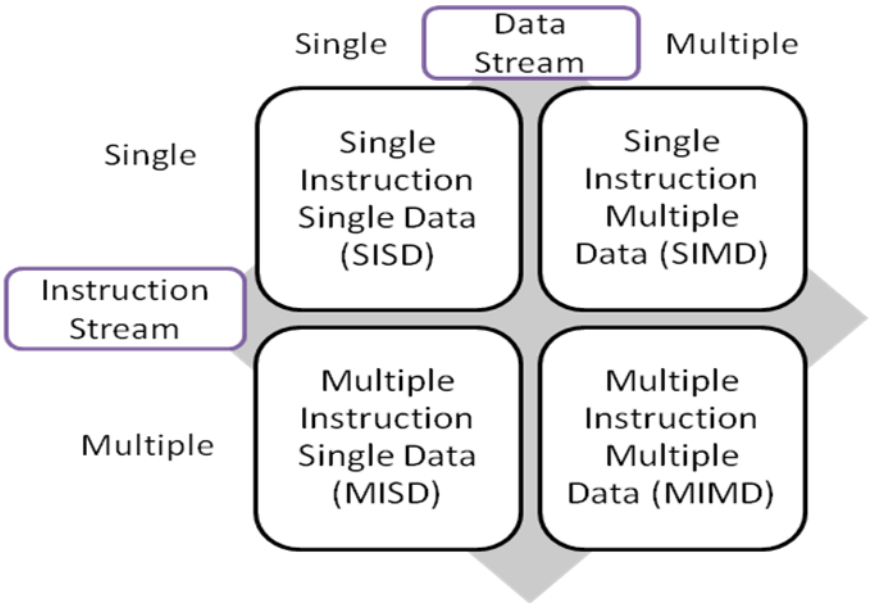

Figura 3 Taxonometria de Flynn para arquitecturas de computador. Adaptado de [30].

2.1.1 Performance de sistemas paralelos

Um dos principais objectivos da computação paralela, consiste em obter uma maior desempenho na computação. Só através da avaliação de desempenho se poderá obter informação relativa às vantagens da paralelização. Torna-se oportuno referir métricas que permitam avaliar este aumento de desempenho e seus limites.

❖ **Aceleração do desempenho (*Speedup*)**

O speedup (aceleração) (S), calcula-se através do quociente entre o tempo da execução sequencial (Ts) e o tempo de execução paralelo em P processadores (Tp)

$$Speedup = \frac{Tempo\ de\ execução\ sequencial\ (Tn, 1)}{Tempo\ de\ execução\ paralela\ (Tn, p)}$$

S pretende avaliar a eficiência de paralelização efectuada. Espera-se sempre que o speedup seja maior que um embora isso não seja necessariamente verdade, devido por exemplo, ao gasto extra de tempo (overhead) introduzido. Dever-se-á ter em conta que existe sempre código não paralelizável, tal como inicializações e leituras de dados de dispositivos por natureza sequenciais. Separando então o tempo de execução do código necessariamente sequencial (Ts) e o tempo de execução do código paralelizável (Tp) usando p processadores obtém-se:

$$Speedup \leq \frac{Ts + Tp}{Ts + \frac{Tp}{p}}$$

É necessário considerar também o custo extra pelo *overhead* nas execuções paralelas (k):

$$Speedup \leq \frac{Ts + Tp}{Ts + \frac{Tp}{p} + k}$$

❖ **Lei de Amdahl**

A lei de Amdahl assume a tentativa de resolução de problemas com tamanho fixo o mais rápido possível. Fornece o caso mais favorável através da equação anterior e mostra que a partir de certo ponto o *speedup* atinge um máximo por mais processadores que se utilizem (limite). Apenas com o aumento de volume de dados juntamente com o número de processadores o *speedup* pode aumentar.

Analise-se o seguinte caso:

Um conjunto de operações a realizar *n* em que cada uma demora uma unidade de tempo e a fracção de operações que exigem execução sequencial *f*. Então *f*n* é o número de operações sequenciais e *(1-f)*n* é o número máximo de operações paralelizáveis em *p* processadores

$$Speedup\ (Sp) = \frac{tempo\ de\ execução\ com\ 1\ processador\ (T1)}{tempo\ de\ execução\ com\ p\ processadores\ (Tp)}$$

$$Tp = (f * n) + \frac{(1-f) * n}{p}$$

$$Sp = \frac{n}{f * n + (1-f) * \frac{n}{p}} = \frac{1}{f + \frac{1-f}{p}}$$

$$\lim_{p \to \infty} Speedup = \frac{1}{f}$$

Se f= 0.05, então o limite máximo do *speedup* será 20. Para obtermos um *speedup* de 100 o f teria de ser inferior a 0.01. Esta observação é conhecida como a lei de Amdahl [5]. Tal como se referiu anteriormente, existe sempre código não

paralelizável, portanto atinge-se inevitavelmente um limite de performance, mesmo aumentando infindavelmente o número de processadores. O gráfico da figura 4 permite visualizar
este efeito com $f = 0.05$.

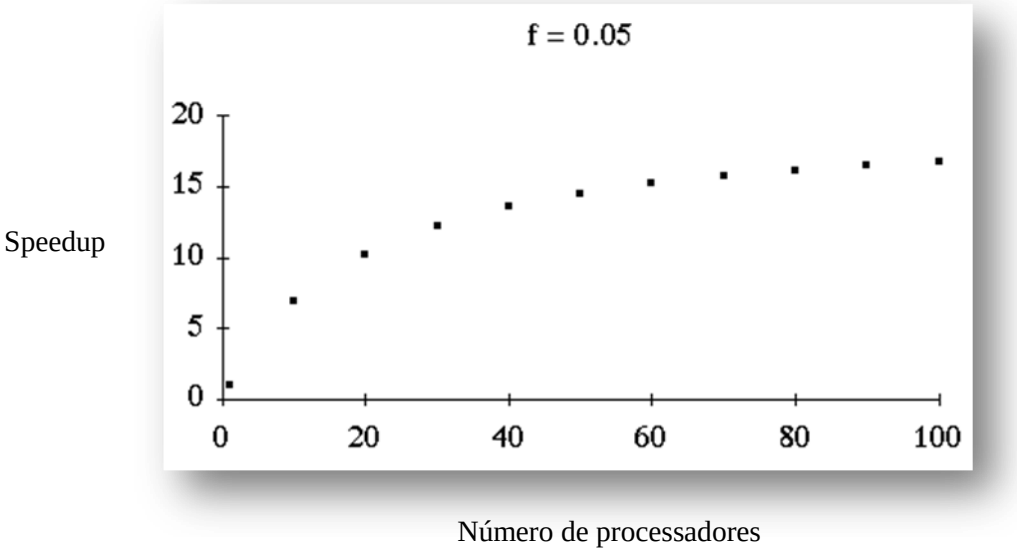

Figura 4: Gráfico da relação entre o *speedup* e o número de processadores

A aplicabilidade da presente lei ao tema desta tese consiste numa advertência com carácter pragmático: a paralelização de aplicações (ou parte delas) faz apenas sentido quando estas possuem algoritmos pesados e estão encarregadas de processar grande volume de dados. Aplicações consideradas 'leves' rapidamente atingem um limite de velocidade com o aumento de processamento É portanto nas aplicações consideradas "pesadas" que vale a pena focar e tirar grandes vantagens na velocidade acrescida com o aumento do número de processadores a executar tarefas. Algumas das fases de processamento de dados tomográficos enquadram-se nesta situação.

❖ **Arquitecturas MIMD de memória partilhada**

Num modelo de memória partilhada os processadores comunicam através de leituras e escritas em memórias partilhadas, igualmente acessíveis a todos os processadores [7], à totalidade de espaço de endereçamento. Todos os processadores têm acesso ao total endereço de memória disponível através de um bus comum (ou outro tipo de dispositivo de interligação) como mostra a figura 5.

Existem dois modelos principais de memória partilhada:

- *Uniform Memory Access* (UMA) – Nesta arquitectura, todos os processadores têm o mesmo tempo de acesso a toda a memória. Tem vantagens pela partilha igual de recursos mas pode ser desvantajoso em termos de escabilidade, uma vez que é difícil garantir esta uniformidade com um nº muito grande de CPUs.

- *Non-Uniform Memory Access* (NUMA) – Neste caso o tempo de acesso a diferentes zonas de memória pelos vários CPUs não é constante. Isto deve-se ao uso de estruturas de interligação que permite usar um maior nº de CPUs e módulos de memória.

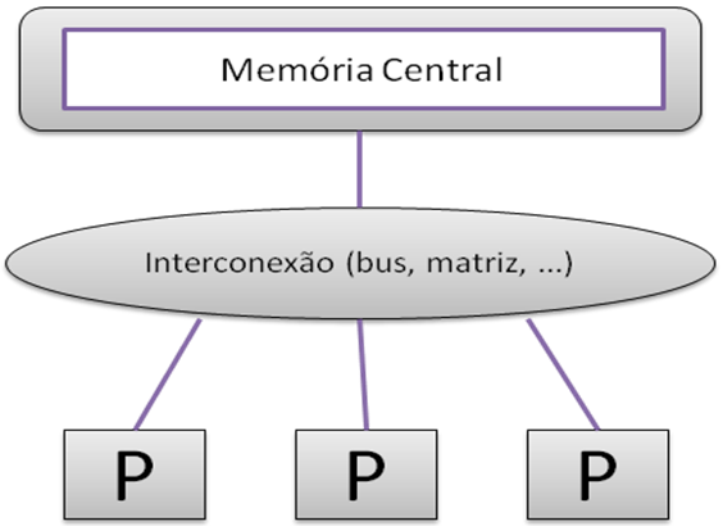

Figura 5: Arquitectura de um sistema de memória partilhada

❖ Arquitectura de SMPs (shared memory parallel computers)

Originalmente o termo SMP designava "sistemas simétricos de multi-processadores" (*symmetric multiprocessor*). Nestes sistemas cada processador partilha memória de forma a que cada um deles aceda a qualquer local de memória com a mesma velocidade [9]. Esta designação corresponde ao modelo UMA já referido atrás. Esta classe de máquinas assume particular importância nos dias de hoje uma vez que muitas das máquinas de secretária multicore hoje disponíveis se podem enquadrar nesta categoria. Máquinas com grande número de processadores são quase sempre NUMA.

A computação de um programa pressupõe o processamento de conjuntos de instruções. Cada conjunto, juntamente com o seu espaço de endereçamento virtual e estado, tem o nome de *thread*. As *threads* são muito comuns a nível de software, mas também são um modelo de hardware. No modelo de hardware comum uma *thread* não tem dependências explícitas com instruções de outras *threads*, mas podem existir dependências implícitas em operações no mesmo endereço de memória [10]. Existem processadores capazes de computar diferentes *threads* em simultâneo. Um processador que suporte essa capacidade é definido como tendo uma arquitectura *simultaneous multi-threading* (SMT). Nos dias de hoje, esta constitui uma característica comum na generalidade dos processadores recentes, mesmo os usados em computadores pessoais e portáteis. Também é possível uma arquitectura com apenas um processador executar várias *threads*, em que o processador vai alternando o uso dos seus ciclos de relógio entre as várias threads conforme o tempo e necessidade.

Nos anos 80 foram construídos computadores com processadores independentes e com uma memória comum partilhada que se tornaram populares até aos dias de hoje. O poder dos processadores foi sempre crescente pelo aumento de transístores num chip e pelo tamanho decrescente dos seus componentes fornecendo sempre maior quantidade de ciclos de relógio. Existem contudo limitações a este crescimento desenfreado pois os transístores não poderão diminuir de tamanho eternamente ao mesmo ritmo. Para ultrapassar estas limitações, o uso de múltiplos processadores voltou recentemente a ter bastante relevância e mostrou-se uma boa alternativa para o

crescimento do poder de processamento. Actualmente, os casos de uso geral usam dois ou quatro processadores com partilha de memória denominados como *multicore*. Haviam sido também criados anteriormente processadores com componentes lógicos replicados obtendo resultados consistentes com a arquitectura de memória partilhada. Estes sistemas são normalmente denominados como *multithreaded* e apresentam baixo custo. Os SMPs dominam o mercado de servidores e são utilizados muitas vezes também como blocos para construir sistemas maiores [11].

❖ **Arquitecturas MIMD de memória distribuída**

Estas arquitecturas distinguem-se das arquitecturas de memória partilhada, principalmente porque a memória é exclusiva a cada máquina. As diferentes unidades de processamento, não comunicam pela partilha de espaços de endereçamento, mas sim por troca de mensagens com cópia de dados entre os espaços de endereçamento dos processos envolvidos. Para a comunicação de dados e obtenção de resultados, impõe-se que as máquinas se encontrem interligadas através de uma rede interna, de preferência de elevada velocidade. A este sistema constituído pelos vários componentes, designa-se normalmente como multi-processadores de memória distribuída ou multi-computadores. É comum também o termo *cluster* (agregado) estado este normalmente reservado a situações em que cada conjunto (processador, memória, periféricos) é construído a partir de hardware de uso corrente, por exemplo o uso de computadores pessoais.

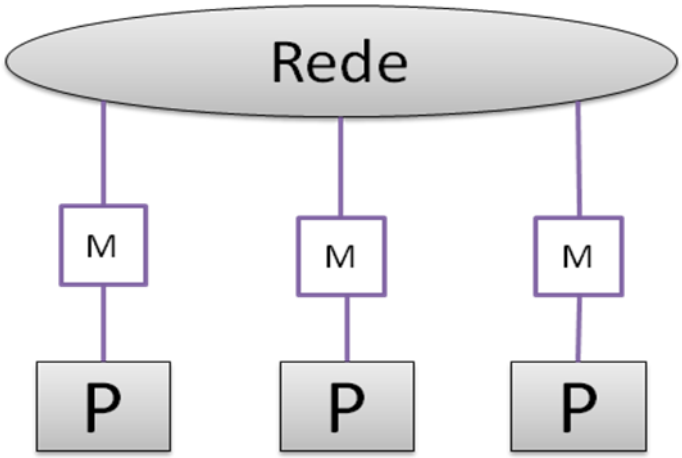

Figura 6: Arquitectura multi-processador de memória distribuída.

2.2 Paralelização de aplicações

Normalmente o problema da paralelização apresenta-se sob a forma de um programa sequencial que necessita ser optimizado recorrendo à distribuição da carga por fontes de processamento independentes. A paralelização, na maior parte das vezes é trivial, mas em algumas situações pode requerer grandes alterações de estrutura e por vezes necessita de novos algoritmos para a computação do problema. O processo de paralelização supõe a identificação do trabalho que pode ser realizado em paralelo, e a sua distribuição. Impõe-se portanto o uso de uma metodologia.

2.2.1 Metodologia da paralelização

O modelo segundo Ian Foster [6] representa a computação paralela como um conjunto de tarefas que podem interagir umas com as outras enviando mensagens através de canais. Uma tarefa é um programa com a sua própria memória que envia e recebe valores de dados locais para outras tarefas através das suas portas. Um canal é uma fila de mensagens que liga portas entre tarefas.

Neste modelo facilmente se distingue o acesso a dados locais ou globais, pois os segundos são feitos usando canais e devem ser vistos como mais lentos. Este modelo é útil para a melhor compreensão dos programas paralelos.

Descrevem-se em seguida, de forma sucinta, as fases de algoritmos paralelos de acordo com a metodologia proposta por Ian Foster

- Decomposição

O primeiro passo na paralelização de um programa sequencial, é a sua decomposição após a análise dos algoritmos. Existem modelos que podem ser usados conforme a situação para ajudar neste processo: mestre escravo, decomposição geométrica e decomposição funcional. O processo de decomposição pode dividir computação, dados ou ambos. No caso da tomografia, a paralelização foca-se nos dados, pois a informação é repartida em blocos para serem processados pelo mesmo algoritmo.

- Comunicação

Após a decomposição da computação e/ou dados, é necessário estabelecer a comunicação entre as tarefas paralelas. Este pode ser classificado segundo vários critérios fora de designação do destinatário (nº de intervenientes na comunicação, sincronização entre emissor e receptor de mensagens, etc.). A comunicação entre tarefas pode ser vista como *overhead* da computação paralela, pois é desnecessária na computação sequencial e como tal deve ser minimizada o mais possível.

- Aglomeração

Por vezes ao paralelizar um algoritmo, podem obter-se demasiadas tarefas ultrapassando a capacidade de suporte do hardware. Como a comunicação entre tarefas produz *overhead* extra, torna-se frequentemente vantajoso aglomerar tarefas

para eliminação de computação desnecessária. (Torna-se porém desnecessário dividi-las quando uma tarefa fica pendente dos dados de outra.)

- *Mapping*

Corresponde à afectação (mapping) de tarefas aos nós de processamento. Para obter bons resultados na computação paralela impõe-se uma boa distribuição de trabalho, isto é, por exemplo, se os diversos nós tiverem poder de processamento semelhante e as tarefas tamanho idêntico, é conveniente que a cada nó seja distribuído o mesmo número de tarefas.

2.2.2 Bibliotecas para programação de aplicações paralelas

Para programar aplicações paralelas podem ser usadas linguagens dedicadas, mas normalmente o que se faz é estender uma linguagem sequencial convencional com uma biblioteca de suporte à programação paralela. Esta biblioteca é acessível ao programador através de uma API (Application Programming Interface).

As APIs para programação paralelas dividem-se em duas grandes classes: as baseadas em troca de mensagens (ex: MPI e PVM) e as baseadas em partilha de memória (ex: Phreads e OpenMP)

Seguidamente detalham-se as APIs que utilizam o modulo de programação em memória partilhada, uma vez que foram as escolhidas para esta tese. Esta escolha deveu-se ao objectivo de construir uma solução de processamento de dados tomográficos que pudesse ser executada numa estação de trabalho isolada que muitas vezes é apenas o que os investigadores da área de Engenharia de Materiais possuem para realizar as suas análises. Hoje em dia é já muito comum encontrar computadores individuais com arquitectura multi-processador com dois ou mais cores. A tendência será para um grande aumento do número de cores em todos os computadores relevando cada vez mais as vantagens da paralelização por memória partilhada.

2.2.3 Programação em memória partilhada

Os compiladores sempre foram responsáveis pela adaptação de um programa para aproveitar ao máximo o paralelismo interno da máquina. Infelizmente é muito difícil faze-la num computador com múltiplos processadores ou *cores* [9]. Torna-se essencial então a contribuição do programador para fornecer informação, indicando as zonas de código paralelizáveis e outros elementos conducentes à realização do processo.

Segue-se a identificação e caracterização das principais APIs destinadas a desenvolver as aplicações paralelas que permitem ao programador definir as zonas paralelizaveis.

❖ ***POSIX Threads***

Formalmente, um processo computacional é um espaço de endereço virtual de uma ou mais *threads* de controlo [11]. Uma *thread*, que também podemos cognominar de processo leve, constitui um fluxo independente de controlo e define-se pelas seguintes características:

- Existe dentro de um processo
- Usa os seus próprios recursos locais
- Duplica apenas os recursos necessários para que esta exista como executável mantendo-se um processo leve
- Tem o seu próprio fluxo de controlo desde que o processo pai exista e o sistema operativo o suporte
- Pode partilhar recursos com outras *threads* independentes
- Termina se o processo pai termina (ou semelhante).
- É leve pois os recursos necessários à sua existência foram criados com o processo pai, sendo muito pequeno o *overhead* associado à sua gestão.

O uso de *threads* num uni-processador pode ser vantajoso em casos em que o bloqueio de uma *thread* numa operação de entrada/saída não impede as outras de prosseguirem.

As *threads* partilham recursos, incluindo espaço de endereçamento. Para evitar conflitos entre elas pode recorrer-se à sua sincronização através de rotinas específicas.

Um processo tradicional em *Unix* tem apenas uma *thread* que possui sozinha a memória do processo e outros recursos. *Threads* dentro do mesmo processo partilham dados globais (variáveis globais, ficheiros, etc.), mas cada *thread* tem a sua própria pilha, variáveis locais e *program counter*. *Threads* são também conhecidas como processos leves, porque o seu contexto é mais pequeno que o contexto de um processo. Esta característica torna a troca de contexto entre *threads* mais leve que a troca de contexto entre processos [17]. O objectivo da programação com *threads* é distribuir a carga a ser processada a cada um destes processos leves, ficando cada um encarregue da sua parte e comunicando através dos dados globais. No entanto fica ao cargo do programador garantir que os dados não sejam corrompidos, por exemplo com duas *threads* a escreverem no mesmo espaço de memória partilhado. Na figura 7 demonstra-se um processo em *Unix* e como as *threads* se encaixam no seu interior.

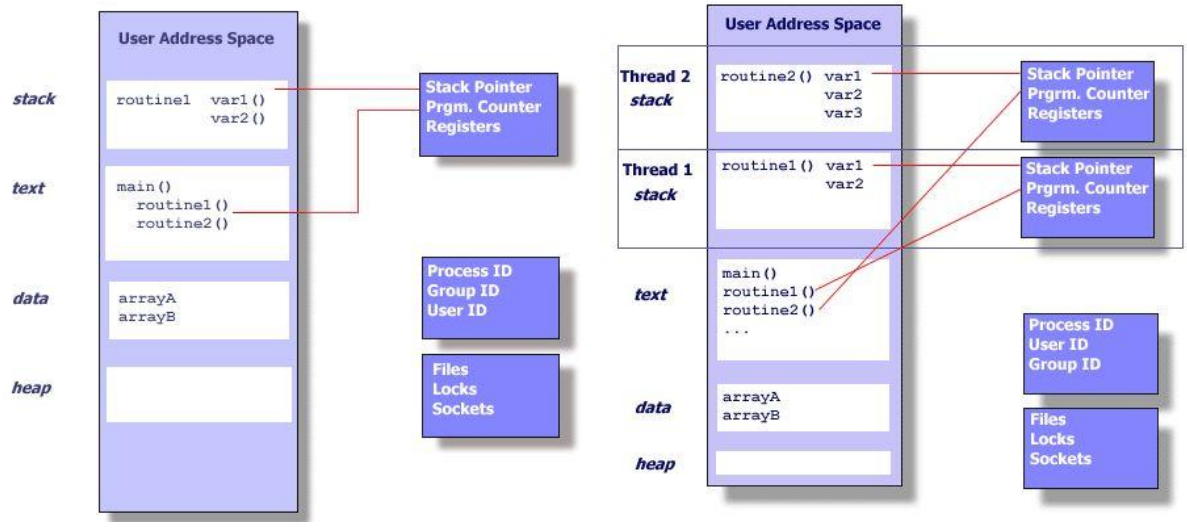

Figura 7: Processo *Unix* e *Threads* dentro de um Processo. Retirado de [28].

Apresenta-se a seguir uma forma possível de paralelizar uma aplicação baseada em processos leves e partilha de memória.

Uma *thread* é criada antes da paralelização do código e é ramificada em várias *threads*, para cada uma trabalhar uma parte do código e todas colaborarem entre si. No final, elas convergem para a *thread* original (a ramificação mestre continua e as outras terminam) obtendo o resultado final como mostra a figura 8.

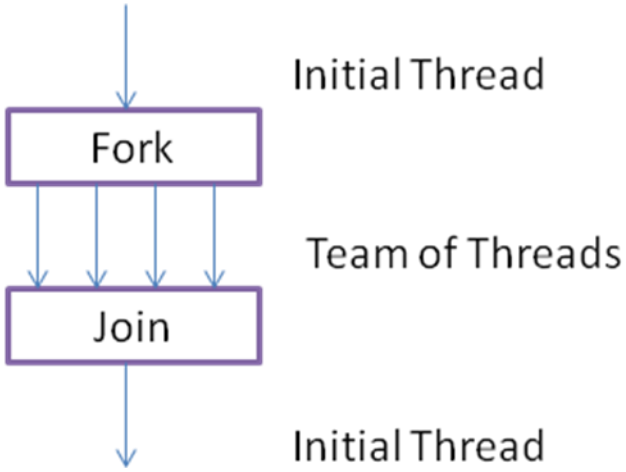

Figura 8: Apresentação esquemática da ramificação de uma *thread* inicial e respectiva união das ramificações. Adaptado de [9].

A biblioteca *POSIX Threads* (*Pthreads*) constitui um conjunto de interfaces para paralelizar código por memória paralela desenvolvida pelo IEEE (*Institute of Electrical and Electronics Engineers*) com o intuito de criar uma interface portátil a vários sistemas operativos *POSIX – Portable Operating System Interface*) [9]. Estas interfaces baseiam-se em funções para manipular threads que são a base da programação paralela por memória partilhada.

• ***POSIX threads* - funções principais**

(1) -Criação de um novo processo leve:

```
int pthread_create(pthread_t *thread, const pthread_attr_t
*attr, void *(*start_routine)(void *), void *arg)
```

(2) -Conclusão de um novo processo leve:

```
void pthread_exit(void *value_ptr)
```

(3) -Espera do processo pai pela conclusão de um processo leve específico:

```
int pthread_join(pthread_t thread, void **value_ptr)
```

Função (1) - é criada uma nova *thread*. Esta thread executa uma função especifica definida como argumento

Função (2) - termina a *thread* e torna o resultado passado como argumento disponível para retorno para mais tarde ser usado na união das *threads*.

Função (3) - é chamada por uma *thread*. Esta fica à espera que outra *thread* específica termine obtendo o valor retornado por esta.

• ***Mutexes* e variáveis de exclusão - principais funções**

As variáveis *Mutual Exclusion* (*Mutex*) permitem sincronizar o acesso a dados para protegê-los no caso de escrita. Nas *POSIX threads* os *mutexes* dão acesso exclusivo a apenas um processo leve, negando acesso aos restantes enquanto este estiver ocupado.

(1) -Inicialização de um *mutex* com atributos específicos:

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutexattr_t *attr)
```

(2) -Destruição de um *mutex* :

```
int pthread_mutex_destroy(pthread_mutex_t *mutex)
```

(3) -Bloqueio de um *mutex* até que este volte a estar disponível outra vez :

```
int pthread_mutex_lock(pthread_mutex_t *mutex)
```

(4) -Desbloqueio de um *mutex*:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex)
```

Função (1) - é inicializada uma variável do tipo *mutex* com atributos disponíveis.

Função (2) - destrói uma variável do tipo *mutex*.

Função (3) - é chamada por uma *thread* para esta obter acesso exclusivo a uma zona partilhada de código que necessite protecção. Se o *mutex* estiver bloqueado, este nega o acesso, caso contrário concede exclusividade à *thread*, e não permite mais nenhum lock até que este volte a ser libertado.

Função (4) - liberta o *mutex* permitindo a outra *thread* obter acesso à zona em exclusão mútua.

Um outro mecanismo de sincronização disponível são as variáveis de condição, que sinalizam threads bloqueadas quando um recurso fica disponível:

• **Variáveis de condição para *Mutexes* - principais funções**

(1) -Inicializar variável de condição:

```
int pthread_cond_init(pthread_condattr_t *attr)
```

(2) -Destruir os atributos de uma variável de condição :

```
int pthread_cond_destroy(pthread_condattr_t *attr)
```

(3) -Desbloquear pelo menos uma *thread* que esteja à espera de sinal :

```
int pthread_cond_signal(pthread_cond_t *cond)
```

(4) -Esperar por uma condição e bloquear o *mutex* até receber sinal :

```
int pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *mutex)
```

Função (1) - inicializa a variável de condição com valores definidos como argumento.

Função (2) - destrói os atributos associados a uma variável de condição.

Função (3) - acorda pelo menos uma das *threads* que esteja bloqueada numa variável de condição específica.

Função (4) – esta função bloqueia na variável de condição específica, que liberta o *mutex* especificado como argumento e provoca que a *thread* que invoca a função seja bloqueada na variável de condição.

❖ **OpenMP**

O *OpenMP* [29] é uma interface de programação de memória partilhada que se destaca por facilitar a programação paralela por memória partilhada. Fornece ferramentas gerais para paralelismo, e o seu uso mais comum é na paralelização de ciclos [12].

Esta API foi definida pelo *Architecture Review Board* (ARB), um grupo de vendedores que uniu forças na última metade dos anos 90 para criar meios de programar uma vasta gama de arquitecturas SMP [9]. Actualmente este *standard* pode ser compilado em *Fortrand*, *C/C++* e praticamente em qualquer arquitectura SMP.

Tal como a programação com POSIX Threads o OpenMP, não constitui uma nova linguagem mas sim uma notação que é acrescida a linguagens de programação existentes e que permite ao programador definir zonas de código e/ou dados que podem ser distribuídos entre os vários processadores existentes. O OpenMP baseia a sua implementação em *threads* tal como as *Pthreads*, mas diferencia-se bastante pelo facto de que em vez de disponibilizar funções para a paralelização de código, disponibiliza *pragmas* para o programador definir zonas de código paralelizáveis e parâmetros de comunicação. O OpenMP fornece instruções adicionais ao compilador (o que não acontece com as Pthreads), para este gerar código paralelo.

O grande sucesso do OpenMP deve-se acima de tudo à sua eficiência na estruturação da programação paralela e no seu fácil uso, tratando o compilador dos detalhes trabalhosos, o que eleva esta API a nível mais alto que os seus antecessores, nomeadamente os PThreads. O OpenMP não possui muitas directivas diferentes, mas as suficientes para cobrir todas as operações necessárias na paralelização do código.

- **OpenMP - *Pragmas* principais**

O OpenMP usa apenas duas construções básicas: *pragmas* e rotinas. Os *pragmas* são directivas disponíveis em Fortran, C e C++ para comunicar informação com compilador. Esta pode ser ignorada mas pode ajudá-lo a optimizar o programa [8]. Normalmente os *pragmas* informam o compilador para paralelizar zonas de código. Todos os *pragmas* do OpenMP começam com "*#pragma omp*". Tal como qualquer *pragma*, estas directivas são ignoradas por qualquer compilador que não suporte estas funcionalidades.

Descrição dos principais *pragmas*:

(1) - Especificação de paralelismo:

```
#pragma omp parallel
```

(2) - Distribuição lógica de *threads*:

```
#pragma omp parallel for
```

(3) - Remoção de barreira de sincronização implícita:

```
#pragma omp for nowait
```

(4) - Adição explícita de barreira de sincronização:

```
#pragma omp for barrier
```

(5) - Barreira de zona crítica:

```
#pragma omp critical [name]
```

(6) - Execução específica para a *master thread*:

```
#pragma omp for single
```

Pragma (1) – Define zona paralela a partir da invocação deste *pragma*. O código é distribuído automaticamente pelo *OpenMP* por várias *threads*.

Pragma (2) – Divide cada iteração do ciclo *for* pelas existentes *threads*. Se for apenas usado o *pragma* (1) num ciclo *for*, todas as *threads* percorrem o ciclo inteiro, provocando bastante redundância na execução do ciclo.

Pragma (3) – No final de cada zona de código paralelo, existe uma barreira implícita para sincronização das *threads* em que cada uma delas pára nesta barreira e não avança enquanto todas não chegarem a este ponto. O mesmo acontece no fim dos *pragmas #pragma omp for*, *#pragma omp single*, e *#pragma omp sections block*. Para remover esta barreira e permitir o avanço independente das *threads*, usa-se este *pragma.*

Pragma (4) – Adiciona uma barreira explícita para sincronização.

Pragma (5) – *Cria* barreira de zona crítica. É possível dar um nome a esta zona de código crítica, no qual apenas uma *thread* pode entrar se não existir outra nesta zona ou outra zona com o mesmo nome. Se não for especificado nenhum nome, é atribuído um automaticamente.

Pragma (6) – Por vezes é desejável que apenas uma thread execute um comando ou zona de código. Este pragma permite definir esta situação.

- ***OpenMP* – rotinas principais**

As rotinas *OpenMP* são usadas primariamente para definir e obter informação sobre o ambiente. Dividem-se em três classes: execução de ambiente, *locks/*sincronização e temporização. Para usar as funções *OpenMP*, o programa deve incluir o ficheiro *opm.h*. Se a aplicação usar apenas *pragmas*, esta inclusão pode se omitida.

As rotinas de afectação (*set*) apenas podem ser invocadas fora de zonas paralelizadas, as restantes podem ser usadas tanto fora como dentro.

Apresentam-se algumas delas:

(1) - Obter número de *threads* em execução:

```
int omp_get_num_threads()
```

(2) -Definir número de *threads* em execução:

```
int omp_set_num_threads(int num_threads)
```

(3) -Bloquear uma zona de código:

```
void omp_set_nest_lock(omp_nest_lock_t *lock)
```

(4) -Desbloquear uma zona de código:

```
void omp_unset_nest_lock(omp_nest_lock_t *lock)
```

Função (1) – Função que permite obter o número de *threads* em execução numa zona de código paralelizada. Caso esta função seja chamada fora de uma zona destas, retorna o valor inteiro 1.

Função (2) – Define o número de *threads* a correr numa zona paralelizável. Por definição o número de *threads* usadas é igual ao número de processadores encontrados no hardware.

Função (3) – As funções para garantir segurança em zonas de código podem substituir o uso dos *pragmas* com o mesmo objectivo. Aconselha-se no entanto o uso dos *pragmas* quando possível, pois estes são muito bem estruturados e fáceis de manipular. A vantagem das funções é a sua flexibilidade, permitindo passar estes *locks* como argumento para outras funções e devem ser usadas nestes casos. Existem dois tipos de *locks*: simples e *nested*. As primeiras permitem o bloqueio apenas uma vez e as segundas guardam o número de vezes bloqueadas. Neste caso trata-se de um *lock nested* e a função é usada para bloquear a zona de código.

Função (4) – Função pertencente ao bloco de funções de *locks nested* que serve para desbloquear uma zona de código. Existem mais três funções para manipular *locks* (*init*, *destroy* e *test*) e existem as versões *nested* e simples para as cinco funções no total. Não se explicitam as restantes neste espaço por serem idênticas aos *mutexes* das *Pthreads*.

2.3 Paralelização de processamento de dados tomográficos

Foram já realizados bastantes trabalhos relacionados com o processamento de dados tomográficos, e respectiva paralelização. Seguidamente referem-se alguns deles:

Um professor francês da Universidade de Bordeaux desenvolveu um software para processamento de imagens tomográficas através da aplicação da micro-tomografia de raios X com radiação do sicrotrão com o objectivo inicial de analisar compósitos de carbono/carbono (C/C) que são bastante usados em aplicações como bocais de foguetes ou travões de aviões cujo mercado está em expansão [18]. Mais tarde a aplicação foi usada também para a caracterização da população de reforços de carbonato de silício [19]. Este programa apresentou-se com capacidade para intervir no tratamento preliminar das imagens necessárias para a representação tridimensional dos compósitos porosos C/C.

Num trabalho anterior a este [1], Paulo Quaresma melhorou o programa através de paralelização das operações mais demoradas (limpeza e estatísticas de granulometria), optimização nos acessos à cache, alterações da escrita e leitura da imagem para reduzir chamadas ao sistema e optimização para percorrer a estrutura em memória principal.

Na optimização paralela realizada, os dados foram separados em blocos com a forma de paralelepípedos. Estes blocos foram processados independentemente em operações de limpeza de imagem, abordagem perfeitamente aceitável. Noutras circunstâncias, como a granulometria, que implica identificar os reforços encontrados nos materiais analisados, poderá gerar resultados indesejados, pois a divisão dos dados em blocos, poderão eventualmente seccionar os reforços.

A paralelização de aplicações tomográficas tornou-se bastante comum, pois trata-se de um processo muito pesado e que consome muitos recursos. Efectivamente existem diversos exemplos da sua aplicação dos quais destaco o trabalho realizado por cientistas no National Center for Microscopy and Image Research, University of California, San Diego que endereçaram a computação de tomografia tanto *on-line* como *off-line.* Os referidos ciêntistas emsaiaram primeiro a exploração da computação paralela em modo *off-line* [20] que consistiu em tratar os dados tomográficos localizados em memória secundária, e através da paralelização, tentar processar a informação o mais rápido possível. Este será também o método escolhido para esta tese. Na Califórnia recorreram ao *Globus*, que é um software para computação em Grid para obter os dados necessários. Apresentaram uma estratégia para o uso de super computadores e *clusters* para melhorar o tempo de execução.

Mais tarde, os referidos cientistas, exploraram a tomografia paralela *on-line* [21] através do mesmo ambiente de programação Grid. A tomografia *on-line* consiste em processar dados à medida que estes vão sendo disponibilizados numa fonte remota. Esta abordagem possui a vantagem de poder mostrar imediatamente a qualidade das imagens processadas e possivelmente a interacção do utilizador sobre o processo. No entanto requer escalonamento ao nível da aplicação e ajustamentos em tempo real pois a disponibilidade dos recursos é dinâmica.

Numa outra universidade, Wane State University Detroit, Michigan foi realizada uma implementação paralela de processamento micro-tomográfico [22], com dados obtidos através da projecção tomográfica de radiação com o formato cónico que necessita de um algoritmo de processamento específico denominado *Feldkamp*. Este algoritmo foi implementado de forma paralela utilizando a divisão de dados por blocos, recorrendo ao *Message Passing Interface* (MPI). Nesta mesma implementação, executada numa rede com 6 *workstations* foram reajustados os fragmentos a distribuir pelos nós para aumentar a performance da computação. Também foi executado num multi-processador simétrico (SMP) com dois processadores, conseguindo obter resultados em quase metade do tempo da versão sequencial.

Outro trabalho realizado por cientistas do CNRS Research Unit UMR em Villeurbanne na França consistiu em comparar algoritmos para reconstrução de dados obtidos pela micro-tomografia com radiação cónica como no exemplo anterior [23], tendo sido analisados vários algoritmos. Com este trabalho descreveram os métodos de paralelização de cada algoritmo tanto para as operações básicas como para as de reconstrução das imagens tomográficas onde podemos verificar que cada algoritmo requer processamento das imagens bidimensionais com ordem de processamento específico condicionando a distribuição de carga entre os vários nós de computação. As implementações destes últimos cientistas foram testadas em várias arquitecturas paralelas. Verificou-se que o algoritmo de *Feldkamp* obteve resultados de forma mais rápida, apesar de as imagens reconstruídas poderem perder alguma qualidade, pois o algoritmo de Feldkamp recorre a algumas aproximações nos dados.

2.4 Visualização

Nos últimos anos os sistemas de visualização científica evoluíram desde uma colecção de aplicações e algoritmos geralmente construídos para uma área específica, suportando tipos de dados muito limitados e requerendo o apoio a tempo inteiro de um programador para sistemas com algoritmos aplicáveis a todas as áreas científicas e de engenharia com suporte a diversos tipos de dados e simples o suficiente para o

investigador poder manuseá-los sozinho [35]. De seguida apresentam-se alguns dos mais relevantes sistemas de visualização existentes.

2.4.1 Sistemas de visualização

❖ **OpenDX**

O OpenDX [36] como refere [37] era um software originalmente designado como IBM Data Explorer. Este produto era totalmente suportado pela IBM e estava disponibilizado para todas as workstations Unix comercializadas. Mais tarde a IBM converteu a última versão disponibilizada comercialmente numa versão open source que pode ser compilada numa grande variedade de plataformas.

Este software foi desenhado especificamente para a visualização de dados, permitindo a visualização tridimensional. O OpenDX permite a visualização de dados em multiplos formatos. Esta é uma grande vantagem, pois simplifica bastante o facto de não ser necessário converte-los num formato nativo, os quais muitas vezes necessitam ser estudados antes.

Existem duas componentes base no OpenDX que são a interface de utilizador e o executivo. A interface de utilizador permite criar programas através de uma interface gráfica seleccionando os subprogramas desejados, designados como módulos. O executivo é um processo separado que manuseia os fluxos de dados, determina a ordem de execução e processa os dados de acordo com as instruções nos módulos. Na figura 9 pode-se observar de forma geral estes componentes no OpenDX.

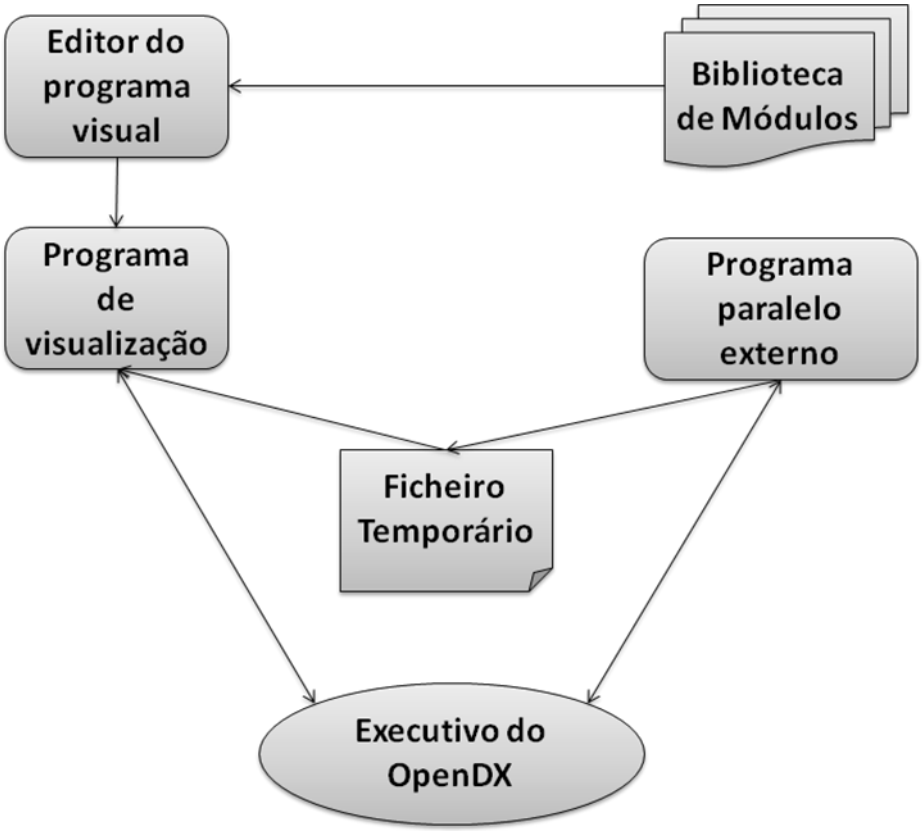

Figura 9: Estrutura do OpenDX. Adaptado de [38].

A componente de interface do utilizador divide-se nas seguintes:

- Um editor gráfico para criar programas visuais
- Conjunto de operações para manipular dados designados como módulos.
- Controlo de execução por parte do utilizador à computação no programa visual

Através do editor gráfico, a criação de programas visuais por parte de não especialistas na área da computação torna-se bastante simples e intuitiva. A construção destes programas é feita num painel com uma lista de módulos, onde o utilizador selecciona as operações que deseja realizar e interliga-as de acordo com o processamento desejado. Na figura 10 pode-se visualizar um exemplo de um programa visual simples com três módulos onde é realizado um processamento simples de limpeza da imagem.

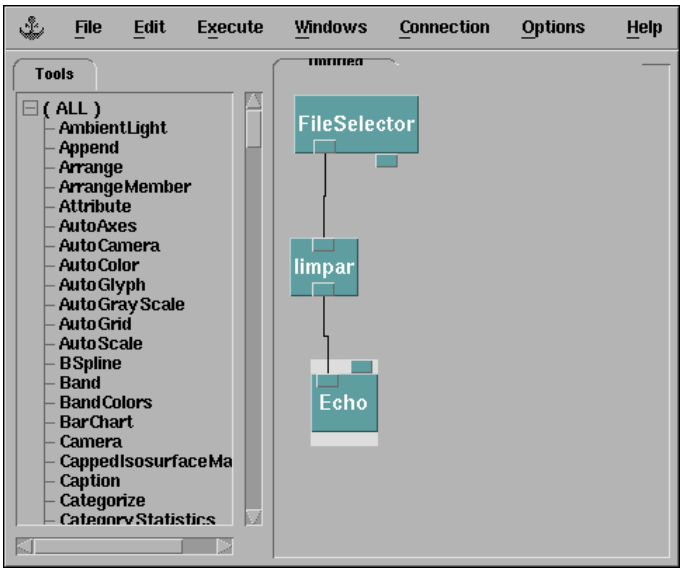

Figura 10: Exemplo de um programa visual no OpenDX.

Como se pode ver pela figura, o programa visual é constituído por módulos que tem o formato de rectângulos e estes estão ligados entre si. Estes são colocados no painel e interligados entre si através do cursor, tudo realizado de forma gráfica. Os módulos possuem entradas e saídas de dados e informação que caminham por fluxos que são as referidas ligações. O produto resultante de um módulo pode ser passado para a entrada de outros. Estes programas visuais podem ser gravados como ficheiros para evitar que seja necessário definir o mesmo programa cada vez que queria executar um processamento.

O OpenDX possui diversos módulos nativos categorizados em bibliotecas, os quais podem ser encontrados na lista na parte esquerda d o editor gráfico de processamento. Também possibilita a criação de módulos para utilizadores mais avançados. Estes módulos podem ser incluídos na lista de módulos nativos do OpenDX e categorizados para integração em programas visuais.

No menu principal do OpenDX é possível também importar dados directamente sem recorrer ao programa visual, bastando apenas definir o tipo de dados. Isto é útil para os casos de visualização simples quando não é necessário realizar processamento sobre os dados.

Existem três tipos de módulos que podem ser criados:

- *Inboard Modules* que são compilados dentro do próprio OpenDX. O executivo do OpenDX é subtituido por outro que inclua os novos módulos dentro.
- *Outboard Modules* correm como processos separados do OpenDX, como tal são *linkados* mais rapidamente pois não é necessário criar uma nova versão do executivo do OpenDX. Contudo se houver uma transferência avolumada de dados, esta será mais lenta nos *outboard modules* pois a comunicação é feita através de *sockets*.
- *Runtime-loadable modules* à semelhança dos *outboard modules*, podem ser adicionados ao OpenDX em qualquer altura mas com a vantagem das transferências de dados entre módulos serem consideravelmente mais leves.

Para a construção de módulos é necessária a criação de 3 ficheiros:

- Um ficheiro com a descrição do módulo e que tenha a extensão .mdf
- Um ficheiro ou mais de código na linguagem C que implementa as operações a realizar.
- Um *makefile* para compilar os módulos desejados.

O OpenDX possui uma ferramenta para facilitar bastante o processo de criação de um módulo. Com esta ferramenta, indicando algumas informações os ficheiros são gerados automaticamente. Na figura 11 pode-se observar a interface da ferramenta para gerar os ficheiros. No ficheiro de código, é gerada a estrutura base, necessitando apenas o programador de inserir o seu código específico para completar o módulo.

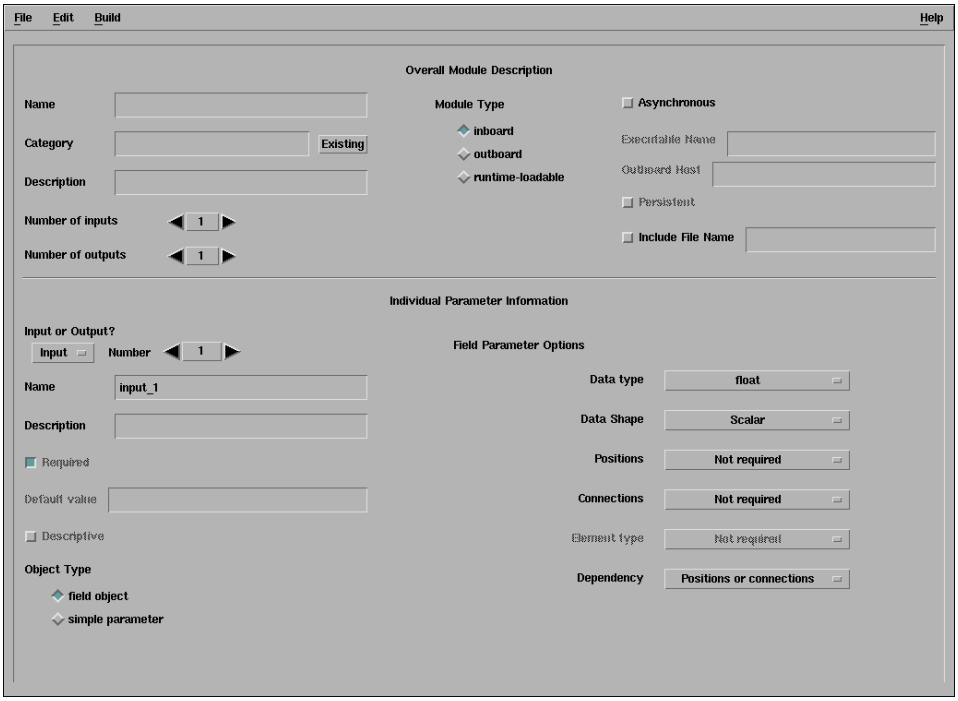

Figura 11: Construtor de módulos do OpenDX.

❖ AVS

O AVS (Application Visualization System) [39] é um software comercial para análise e visualização de dados usando especialmente na área de física e engenharia [40]. Está disponível para as grandes plataformas Unix e mais tarde foi criada uma versão AVS/express para Windows NT e derivados.

Esta plataforma, tal como várias outras, foram construídas à semelhança do OpenDX. Permite a visualização tridimensional de dados com controlo por parte do utilizador e uma interacção simples para utilizadores inexperientes na área da computação com botões, barras e bastante versatilidade. Tal como no OpenDX, é possível integrar vários componentes de computação (módulos) e interliga-los entre si de forma gráfica e intuitiva criando uma rede de processamento de fluxos de dados.

Permite também a programadores criar os seus próprios módulos através de uma *framework* para linguagens C e FORTRAN.

O AVS é constituído por 5 componentes gráficos:

- Visualizador de imagens – Responsável pela visualização e manipulação de imagens bidimensionais
- Visualizador de gráficos – Acessibliza ao utilizador visualizar dados em forma de gráficos (linhas, pontos, etc.)
- Visualizador geométrico – Com a mesma interface do visualizador de imagens, aqui são manipulados os dados tridimensionais. No painel de controlo deste componente são lidos e escritos os dados, são adicionados efeitos à imagem apresentada e são modificados alguns parâmetros da visualização.
- Editor de rede – Este componente é a base do AVS pois especializa-se no processamento dos dados, ao contrário dos anteriores que se dedicam à visualização. É onde o utilizador manipula os módulos, define os seus parâmetros e as respectivas interconexões.
- Aplicações AVS – Este é o menu base do AVS, onde se pode seleccionar para ver visualizar exemplos, para visualizar dados sem ter que editar programas visuais e o próprio editor dos referidos programas visuais.

❖ **Mercury – Avizo**

O Avizo [41] é também um software comercial, desenvolvido pela Mercury Visualization Sciences Group, também vocacionado para a área científica, que permite a visualização de dados de forma tridimensional, e tal como os referidos anteriormente permite construir programas visuais a partir de módulos de forma gráfica e simples.

De seguida apresento algumas características específicas que este software permite:

- Importar quaisquer tipos de dados tal como o OpenDX, inclusivamente dados numéricos referentes a simulações e animações.
- Visualizar diversas imagens em simultâneo em janelas diferentes, ou visualizar a mesma imagem em ângulos diferentes ao mesmo tempo.

- Recortes de imagem interactivamente com recortes ortogonais.
- Processar a imagem em tempo real e mudar a luz incidente conforme necessário
- Interagir com grandes volumes de dados em memória
- Calcular distâncias e posições interactivamente com o rato
- Integração com o matlab
- Exportação de apresentações para vídeos

Este software possui pacotes extras de funcionalidades cujas licenças podem ser adquiridas por custo extra e que se adaptam a diferentes necessidades, tais como pacotes para visualizar dados em ecrã multi-mosaico, trabalhar concorrentemente em equipa, efectuar processamentos de dados específicos a certas áreas, criação de módulos, etc.

O Avizo permite também a inclusão de módulos criados pelo utilizador na linguagem C++ com uma ferramenta de ajuda para estas situações.No entanto é necessário adquirir uma licença para um pacote extra que permite esta funcionalidade.

2.4.2 Visualização de dados tomográficos em paralelização

A visualização de resultados em aplicações tomográficas pode tornar-se extremamente vantajosa. Houve já algum trabalho relacionado neste âmbito, nomeadamente por cientistas em Nova Iorque [24] que criaram uma ferramenta, com o intuito de produzir visualizações paralelas, nomeadamente para dados obtidos na computação tomográfica, para que a informação fosse visualizada em monitores multi-mosaico.

Nesta aplicação foi usado o *rendering* do *Chromium* [31], uma interface para visualizações clusters ligados a monitores multi-mosaico. Realizaram extensões de APIs populares de visualização como o *Open Inventor* [32] e o VTK [33] adicionando algumas funcionalidades às operações de visualização. Por fim integraram a ferramenta *Chromium* com as operações estendidas das APIs de

visualização para que fosse possível visualizar a informação gerada em *clusters* nos referidos monitores multi-mosaico.

Para testar esta implementação, estes investigadores usaram exemplos de dados obtidos através da micro-tomografia, nomeadamente uma de processamento leve com fluxo rápido de dados de superfícies da cabeça de uma barata e uma outra mais pesada com dados tomográficos mais extensos de uma rocha.

Houve também trabalho de outros investigadores da Universidade de Chicago que tirou proveito da visualização [25]. Utilizaram uma ferramenta chamada Volumizer [34] que lhes permitiu processar a visualização de grandes volumes de dados de forma rápida por hardware. Permitiu ainda interacção e visualização dos resultados em tempo real. Esta experiência apresentou como desvantagem a restrição de aplicação apenas a arquitecturas compatíveis, devido à sua optimização a hardware específico. A visualização dos dados tridimensional dava a possibilidade de analisar a estrutura dos dados. Esta ferramenta permitiu também, sob a forma de *stream* vídeo, o envio da visualização dos dados para computadores remotos. Um outro trabalho que recorreu ao uso da visualização tomográfica, pertenceu a uma tese de mestrado no Brasil [26]. Incidiu na análise de imagens de solos que eram obtidas a partir do micro-tomógrafo. As imagens bidimensionais foram tratadas com as transformações inversas de *Fourier* e posteriormente transformadas em tridimensionais pela interpolação através da função *B-Wavelets* encontrada na linguagem VRML.

Diversos projectos científicos fizeram uso da ferramenta OpenDX para processamento e visualização de dados de forma tridimensional. Um desses exemplos foi na análise biomecânica da actuação de forças no sistema muscular e esqueletico [42]. O objectivo desses estudos foi introduzir um novo sistema para captar a geometria de pacientes individuais, permitindo calcular a força dos músculos e dos pontos de junção dos ossos. O sistema foi construído com base em imagens tomográficas obtidas dos ossos dos indivíduos. As imagens sofreram alguns processamentos e foram por fim visualizadas recorrendo ao OpenDX.

Noutro exemplo foi criada uma ferramenta para visualizar dados de caminhos metabólicos e para avaliar quais são os mais afectados por mudanças nas

experiencias genéticas. [43] A análise é implementada com ferramentas de estatística para análise de dados de expressão de genomas para filtrar os caminhos de genes que realmente interessam. Estes são posteriormente visualizados com o OpenDX permitindo uma examinação detalhada dos níveis de expressão observados nas experiencias de acordo com os caminhos de genomas.

2.5 Balanço

Na pesquisa bibliográfica realizada não se encontrou um sistema de processamento de dados tomográficos que oferecesse, em conjunto, as funcionalidades de suporte de execução paralela, flexibilidade, facilidade de utilização por não especialistas de informática e visualização de imagens tridimensionais.

Nos capítulos seguintes descrevem-se os trabalhos que, para elaboração desta tese foram realizados, direccionados para a construção de um sistema que superasse as lacunas mencionadas.

3 Arquitectura da solução proposta

3.1 Organização Geral

A organização da estrutura foi moldada para usufruir totalmente do OpenDX. Com o OpenDX o utilizador pode interagir facilmente com o sistema através do uso de módulos que são sub-programas que em conjunto formam um programa completo composto pelas operações desejadas. Através destes, constroem-se programas visuais com interligações de fluxos de dados onde o utilizador pode decidir que processamentos são aplicados aos dados, a ordem dos mesmos e o valor de parâmetros usados num dado passo.

As operações de tomografia encontram-se organizadas e divididas por módulos. O OpenDX possui de base, uma grande quantidade de módulos para diversos fins, mas possui também uma framework para a construção de novos módulos; esta framework suporta a programação em C para o utilizador inserir o código específico que lhe interessa. É permitido nos módulos a invocação de programas externos, inclusivamente que estes recorram a execução paralela (OpenMP, MPI, etc.). Nada impede que estes programas corram numa máquina paralela remota.

Cada operação do Tritom recebe um ficheiro inicial e retorna outro com o resultado das alterações efectuadas, sendo este o método de troca de dados e informação entre os vários processamentos e passos de execução. A passagem dos dados por ficheiros torna o programa mais lento, mas é indispensável guardar os resultados intermédios para processamentos ou visualizações posteriores.

As operações foram transformadas em sub-programas independentes e são chamados pelos módulos formando uma cadeia de processamento. Os módulos acabam por se

reflectir numa ponte entre o OpenDX e os processamentos do tritom. A razão da invocação externa das operações é pelo facto de algumas delas recorrem ao OpenMP necessitando uma compilação especial para esse efeito. Com sub-programas externos é possível a execução paralela dessas operações. Na figura 12 pode-se analisar a organização da solução.

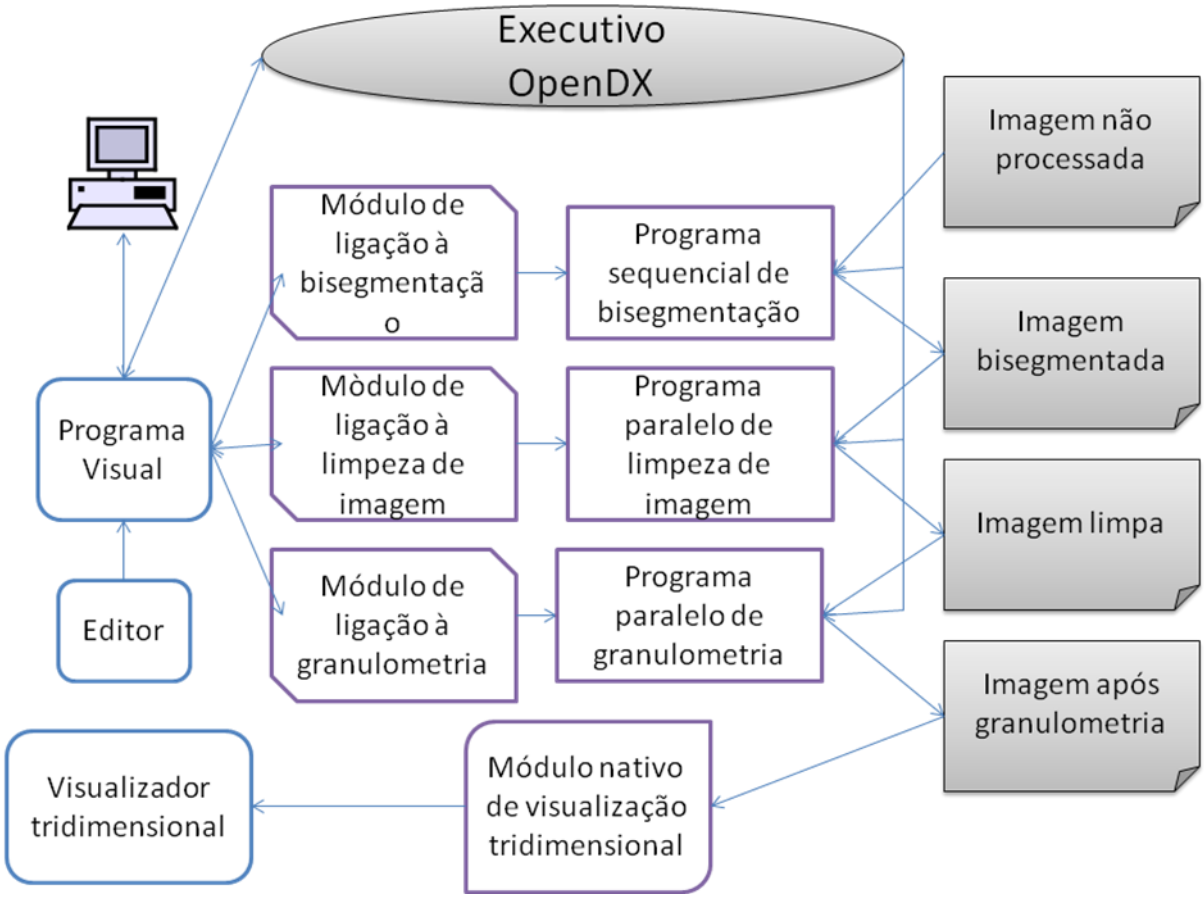

Figura 12: Esquema do Tritom integrado no OpenDX.

3.2 Módulos desenvolvidos

Como já referido o OpenDX possui diversos módulos categorizados mas permite o desenvolvimento de novos módulos à medida das necessidades do utilizador.

A partir das várias operações existentes no Tritom, foram criados módulos para suportar a integração desses processamentos dentro do OpenDX. Os módulos criados usam a framework em C disponibilizada e são bastante simples. Têm definido os parâmetros de entrada e saída que em quase todos se resume ao ficheiro de dados, o ficheiro de retorno com os dados processados e mais alguns parâmetros extra de informação. Os valores de entrada e saída são então passados como argumento a um programa externo, que foi extraído do Tritom e corresponde a uma das operações disponíveis para processamento tomográfico. Este programa externo é invocado a partir da função system disponível no C para executar comandos shell em linux e que retorna no final da execução do comando. As mensagens que os sub-programas retornam são todas passadas ao OpenDX e podem ser visualizadas numa janela de controlo de execução para o utilizador avaliar a situação e progresso de execução.

O OpenDX permite a construção de módulos que são programas paralelos. Uma das formas de o atingir é com funções existentes na biblioteca do OpenDX para esse efeito que permitem distribuir trabalho pelos processadores disponíveis. Outra é executar aplicações externas que por si já sejam paralelas, este foi o método usado, tanto em módulos paralelos como em sequenciais. A razão da escolha deste método foi poder usar a biblioteca OpenMP em subtituição da fornecida pelo OpenDX, pois foi a estudada nesta tese e aplicada no tritom e cujas vantagens foram referidas em capítulos anteriores. Outra vantagem de invocar aplicações externas é a possibilidade dessas aplicações serem executadas remotamente

Algumas operações foram criadas em acrescento para o tritom de acordo com as necessidades reveladas pelos investigadores de materiais. Estas foram implementadas também em módulos para o OpenDX e são as seguintes:

- Limpeza de fundo – Conversão de todos os vóxeis que não estejam incluídos numa zona considerada partícula em branco. numa zona considerada partícula em branco.
- Conversão de ficheiros em formato vol (1 real por ponto no espaço 3D) para o formato usado no Tritom (1 byte por ponto)

- Recorte de partículas – Criação de sub-ficheiros tomográficos de acordo com partículas detectadas
- Gerador de imagens bidimensionais no formato pgm – Recorte de um conjunto de imagens num intervalo definido pelo utilizador
- Fusão de imagens tiff para criação de um ficheiro tomográfico – Leitura de imagens tiff sequenciais gerando um ficheiro tomográfico no formato usado pelo Tritom

O trabalho desenvolvido no âmbito desta tese visa o tratamento de imagens tomográficas de materiais compositos em que sobre uma matriz metálica se difundem reforços em partículas cerâmicas.

3.3 Paralelização usando OpenMP

As duas operações mais pesadas do tritom foram paralelizadas através de memória partilhada para melhorar consideravelmente os seus tempos de execução em arquitecturas multi-processador. Estas operações são respectivamente limpeza de imagem e identificação de partículas. Nas imagens tomográficas essas partículas, correspondem a aglomerados contiguos de vóxeis da mesma cor, conhecidos por blobs.

A abordagem usada para a paralelização destas operações foi a paralelização geométrica introduzida no capítulo 1 desta tese. A imagem, como pode ser visto pela figura 2, é dividida blocos com a forma de paralelepípedos conforme o número de processos e estes possuem dimensão idêntica.

A paralelização da operação de limpeza é relativamente simples, pois o processamento de cada um dos blocos é independente, não existindo necessidade de trocar informação entre os processos. Esta situação corresponde a uma situação conhecida por “embaraçosamente paralela” em que para um tempo de execução sequencial T, utilizando P processadores o novo tempo é reduzido para T/P.

Na operação de detecção de partículas denominada como granulometria a situação complica-se pois um blob pode encontrar-se repartido entre blocos como se pode ver pela figura 13. Nestas situações é necessário intercâmbio de informação entre blocos. Nesta tese procedeu-se à partilha de bloco6s entre dois processos (entre o processo responsável pelo bloco e seus vizinhos) em que quando um processo está a tratar um blob, se este estiver também na zona do bloco vizinho, ele invade a vizinhança estritamente para acabar o processamento do blob.

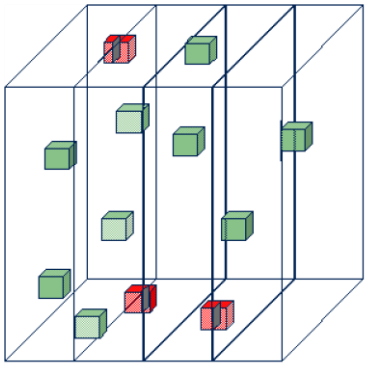

Figura 13: Alguns blobs estão entre dois blocos e necessitam partilha de informação. Retirado de [38].

3.4 Balanço

Descreveu-se neste capítulo a organização geral da solução proposta, que cumpre os objectivos inicialmente propostos. Como o sistema concebido e implementado incluí a paralelização das operações mais demoradas e está integrado no OpenDX. A estrutura base do Tritom, como pode ser visto, foi alterada com esta integração e as operações passaram a ser invocadas por módulos que o utilizador recorre graficamente na interface do editor de programas do OpenDX. Mencionaram-se também as novas operações disponíveis no software.

No capítulo seguinte descreve-se a implementação a fundo do trabalho realizado nesta tese. Descrevem-se as operações disponíveis no Tritom com especial relevo nas que foram paralelizadas e nas criadas nesta tese. Descreve-se a paralelização realizada e respectivos testes de tempo de execução para análise de eficácia deste procedimento. Por fim demonstra-se um exemplo de aplicação do Tritom na análise de um micro esfera cerâmica contida numa espuma sintáctica.

4 Realização da solução proposta

Neste capítulo, faz-se a descrição detalhada de cada um dos componentes da solução apresentados no capítulo 3. Começa-se pela forma como se integram os módulos de processamento no OpenDX, seguindo-se das descrições dos módulos que foram adaptados e parte da versão sequencial do Tritom. Prossegue-se descrevendo os módulos correspondentes e os tratamentos que foram paralelizados, apresentando-se resultados de tempos de execução para um mono-processador com 2 cores e um multiprocessador com 16 cores. Apresenta-se no final um exemplo de uso do Tritom aplicado a uma partícula de cerâmica numa espuma sintáctica.

4.1 Módulos sequenciais

Esta secção descreve os módulos sequenciais começando por referir a forma de os integrar no OpenDX.

4.1.1 Integração no OpenDX

O Tritom foi integrado com o OpenDX, que como referido anteriormente permite a visualização de dados de forma tridimensional e ainda uma fácil interacção para um não especialista na área da computação.

O Tritom na fase anterior à integração estava construído para ser executado em modo consola com uma lista de comandos para o utilizador pouco intuitiva e em que a ordem de operações era rígida obrigando a seguir um alinhamento. Os ficheiros de

código em C estavam separados praticamente por operações e interligados pelo ficheiro principal do programa responsável pela interacção com o utilizador.

Com vista a integrar o tritom no OpenDX, o programa foi desmembrado em sub-programas independentes que são executados só por si sem nenhuma dependência de restante código. Além de motivações relacionadas com as prácticas usadas no desenvolvimento de grandes programas, cada módulo no OpenDX tem que ser autónomo e por si só constituir um programa que por sua vez comunica por fluxos de dados para os restantes.

Foram criados módulos para cada uma das operações disponíveis no Tritom de acordo com a framework específica para a sua criação, que requer a utilização da linguagem de programação C. Para a criação de módulos dentro da referida framework impõe-se a introdução do código específico das operações a realizar. No trabalho que aqui se apresenta, optou-se por introduzir na framework apenas invocações às sub-aplicações externas correspondentes às operações extraídas do Tritom. A decisão em recorrer a esta metodologia impôs-se face ao objectivo de paralelizar o Tritom com recurso ao OpenMP, que necessita de uma versão de compilador específica não suportada pelo OpenDX. Salienta-se que uma das vantagens de invocar externamente as operações a partir dos módulos é o facto de estas poderem ser executadas remotamente em outras máquinas fornecendo uma maior liberdade de execução.

4.1.2 Módulos adaptados do tritom

Seguidamente descrevem-se sumariamente as funcionalidades fornecidas por cada módulo sequencial existente previamente no tritom:

Calcular Histograma e sua derivada:

Esta é a operação para a construção do histograma, percorrendo os dados de toda a imagem e calculando a frequência de dados desta. Os valores extremos de frequência

são o 0 (zero) para o preto e o 255 para o branco. A frequência de uma cor pode ser calculada pelo número de ocorrências *n* dessa mesma cor sobre o número total de vóxeis *v* existentes na imagem.

$$f = \frac{n\ ocorrencias\ da\ cor}{v\ vóxeis}$$

A derivada do histograma constitui a diferença entre frequências de cor consecutivas.

$$f' = f(cor\ c+1) - f(cor\ c)$$

A operação produz um ficheiro com a extensão *histo* para o histograma e *histoder* para a derivada do histograma. O tempo de execução desta função é pequeno e como tal não necessita paralelização. O algoritmo do histograma é então computado da seguinte forma:

```
for cada vóxel da imagem do
if cor de vóxel = a
then vector[a]= vector[a]+1
for cor = 0 to cor = 255 do
vector[cor] = (vector[cor+1] - vector[cor]) / total de vóxeis *100
```

Bisegmentação (Preto/Cinzento/Branco):

Esta operação separa a imagem em apenas três cores: branco, preto e cinzento. É solicitado ao utilizador que apresente um valor mínimo (*vmin*) e um valor máximo (*vmax*) para os tons de cinzento, para referência. Todos os valores inferiores a *vmin* encontrados na imagem são considerados preto e os voxeis ficam com valor 0 e todos os superiores a *vmax* são considerados brancos e são afectados ao valor 255. Os valores considerados de cor cinzenta são unificados num único tom de cinzento cujo valor é 127. No final da execução é produzido um ficheiro de extensão *bi-segmented* com as modificações geradas na imagem. Esta função foi optimizada por Paulo Quaresma percorrendo o ficheiro de dados sequencialmente na leitura. O seu tempo

de execução é pequeno e não necessita paralelização O algoritmo desta operação, é definido da seguinte forma:

```
for cada vóxel da imagem do
if cor de vóxel < vmin
then cor de vóxel = preto
else if cor de vóxel > vmax
then cor de vóxel = branco
else cor de vóxel = cinzento
```

Dilatação do preto e branco dentro do cinzento (histerese):

Função para dilatar a cor preta e branca dentro do cinzento. Em cada vóxel é avaliada a vizinhança. Caso a predominância seja o preto, o vóxel é pintado de preto. O mesmo procedimento acontece pintando o vóxel de branco caso a vizinhança seja predominantemente branca. Caso haja o mesmo número de vóxeis pretos e brancos vizinhos, então é gerado um número aleatório para decidir entre o branco e o preto. A função retorna no final o número de vóxeis tratados e cria um ficheiro de extensão *clean_h*. O processamento é rápido e não necessita paralelização.

```
for cada vóxel cinzento do
if cor vizinhança predominante = branco
then cor vóxel = branco
else if cor vizinhança predominante = preto
then cor vóxel = preto
else cor vóxel = aleatório entre branco e preto
```

Colorir em P ou B os blobs cinzentos (conforme a cor da pele) (invasão por percolação):

Esta função marca o interior e a pele de um blob cinzento. Um blob apresenta-se como uma área de vóxeis que potencialmente podem constituir uma partícula a analisar. A função determina uma *bounding box* que contem o blob e de seguida pinku ta os vóxeis da cor predominante na pele do blob. A cor predominante é calculada a partir dos 6 vóxeis vizinhos no espaço tridimensional. É criado um ficheiro de extensão *inv_perco* apos execução. A operação é curta e também não necessita paralelização. O algoritmo é o seguinte:

```
for cada vóxel cinzento do
for vizinho = 1 to vizinho = 6 do
determina cor predominante da pele
pinta vóxel de acordo com a cor predominante da pele
```

Colorir em P ou B os blobs cinzentos (cor unica):

Nesta função o utilizador decide se quer transformar os vóxeis cinzentos restantes em brancos ou pretos e são todos convertidos para essa cor. A função deve ser usada após a percolação e produz um ficheiro de extensão *PB*. Devido à rapidez de execução também não foi paralelizada:

```
for each vóxel cinzento do
pintar de branco ou preto
```

Percolação (branco = poros) e eliminação da porosidade fechada:

Esta operação executa após limpeza de imagem ou histerese. Analisa os vóxeis brancos para detectar poros e eliminar os fechados. A função determina os poros

através de uma vizinhança de 26 vóxeis no espaço tridimensional. Não necessita paralelização devido à sua curta duração.

Aplicar imagens-máscaras:

Esta operação usa a imagem original e a imagem binária onde apenas existem vóxeis brancos e pretos. O resultado constitui a imagem original, com os vóxeis convertidos para branco correspondentes aos vóxeis brancos encontrados na imagem binária. Não necessita paralelização.

4.1.3 Novos módulos criados

Foram também introduzidas novas funcionalidades no tritom e consequentes módulos conforme as necessidades dos investigadores na área de Materiais. Em seguida apresento uma breve descrição sobre essas funcionalidades:

Limpeza de fundo: Esta operação é realizada após a granulometria pois necessita dos resultados provenientes desta operação. Através dos ficheiros onde estão descritos os blobs encontrados na imagem tomográfica, a limpeza de fundo limpa a área total que não for considerada pertencente a um blob, convertendo os vóxeis para o valor 255 (branco).

```
for cada vóxel cinzento do
if voxel ∉ particula
then cor vóxel = branco
```

Conversão de ficheiros vol: Os ficheiros tomográficos que os cientistas da área de materiais possuem, têm a extensão .vol e cada *vóxel* é representado por quatro bytes. O tritom foi construído para receber ficheiros cujos *vóxeis* estão representados por um byte. Assim sendo foi criada esta operação para realizar a conversão dos vóxeis de 4 bytes para 1 byte, permitindo assim a imagem tomográfica ser processada normalmente pelo tritom. Esta conversão não causa perdas significativas de qualidade na imagem.

Recorte de partículas: Numa imagem tomográfica, normalmente são encontradas diversas partículas através da granulometria e muitas vezes algumas partículas encontradas são apenas ruído na imagem mas com volume suficiente para ser considerado como partículas normais. Torna-se difícil analisar as partículas numa imagem grande com ruído e várias outras partículas à volta, como tal foi criada esta operação para que de acordo com os resultados obtidos na granulometria, seccione a imagem tomográfica de acordo com os blobs encontrados. São gerados ficheiros independentes por cada partícula encontrada para que cada uma possa ser analisada convenientemente. Com a ajuda de um módulo nativo ao OpenDX "Sequencer" é possível visualizar sequencialmente todas as partículas detectadas para facilmente observar as que realmente interessam.

```
for each partícula do
get(partícula_info)
recorte = recortar(partícula_info)
newFile = recorte
```

Gerador de imagens de recorte no formato pgm: Foi criada uma operação para dada uma imagem tomográfica, gerar recortes bidimensionais da imagem. Os recortes são feitos ao

longo da coordenada z da imagem e são gerados de acordo com um intervalo de valores de z definido pelo utilizador.

Fusão de imagens de extensão tiff num único ficheiro tomográfico: Algumas das amostras que os cientistas da área de materiais possuíam, estavam apenas disponíveis em conjunto de imagens tiff. Foi necessário criar um processamento que permitisse ler um conjunto de imagens pertencentes a uma amostra, e as convertesse num único ficheiro tomográfico para que se pudessem seguidamente realizar as operações normais desejadas.

4.2 Paralelização das operações de limpeza e granulometria com OpenMP

Procedeu-se à paralelização de operações do programa Tritom recorrendo à API *OpenMP*, para acelerar a sua execução em ambientes de memória partilhada.

A estratégia de paralelização foi a mesma seguida por Paulo Quaresma [1], ou seja, paralelização geométrica. Neste método, o volume de dados é dividido em blocos. Cada processo leve ficará responsável por tratar um conjunto de blocos em paralelo.

4.2.1 Limpeza de imagem

A operação de limpeza de imagem é normalmente usada após a bisegmentação que normalmente é constituída por secções brancas e pretas com fronteiras cinzentas. É frequente encontrar vóxeis brancos em regiões pretas e vice-versa, estes vóxeis são considerados como ruído na imagem e necessitam ser eliminados. A limpeza de imagem é responsável por este tratamento que é muito demorado, como tal necessita ser paralelizado e de seguida refere-se este processo.

A operação de limpeza deste programa foi paralelizada no presente trabalho através do OpenMP. Os dados tomográficos, de acordo com este processo, cada vez que a função executa, estes são separados em blocos. São divididos de acordo com o eixo

do X que é a ordem encontrada no ficheiro de dados para um processamento sequencial mais eficaz como demonstra a figura 14. Cada bloco é processado por um processo distinto. Os dados são processados contiguamente por cada processo com vantagens evidentes no uso ciclos dos CPUs.

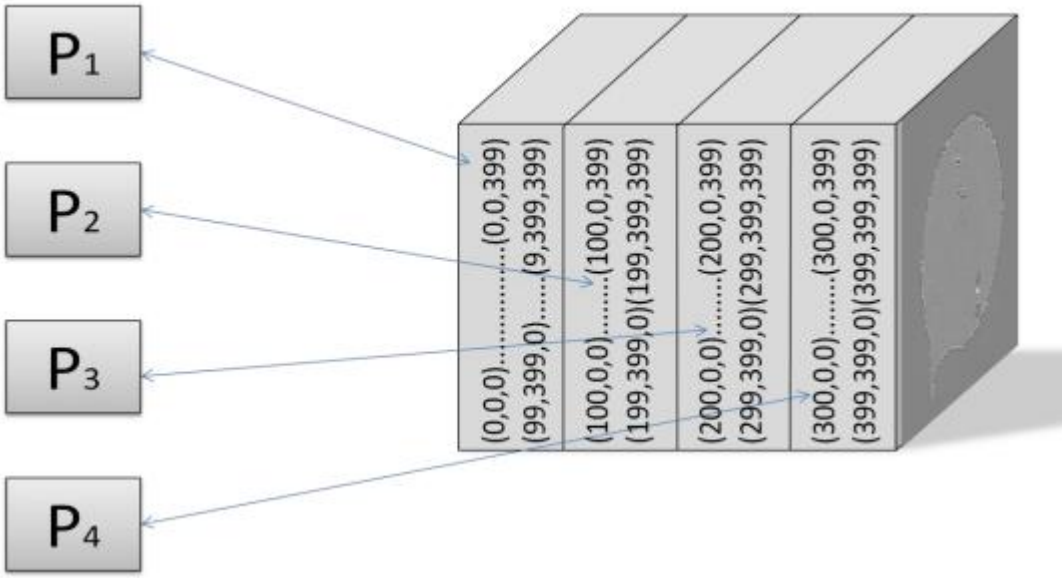

Figura 14: Distribuição dos dados tomográficos, separando-os de acordo com a coordenada X. Assume-se na figura o processamento de dados tomográficos de dimensões 400*400*400, com quatro processadores, sendo atribuído a cada processador 100*400*400 dados.

Apresenta-se seguidamente o código relevante da secção paralelizada:

```
(...)
#pragma omp parallel
   nettoie(lf,la,ha,pr,octet_ptr);
(...)
void nettoie(int lf,int la, int ha, int pr, unsigned char*
octet_ptr)
{
(...)
   nthreads = omp_get_num_threads();
   #pragma omp single
   {
         printf("n threads: %d\n", nthreads);
   }
   #pragma omp for
for (i=0;i<lf;i++) {
      printf("Layer no. %d \n",i);
```

```
      for (j=0;j<ha;j++) {
        for (k=0;k<pr;k++) {
       if (octet(octet_ptr,i,j,k,ha,pr) == GRIS) continue;
(...)
```

Foi também realizada a paralelização desta operação através das *POSIX Threads* para obter uma comparação de performance e de estruturação entre as duas ferramentas. A estratégia usada no paralelismo foi a mesma.

```
(...)
pthread_t *th_a;
(...)
th_a = calloc(nthreads,sizeof(pthread_t));

//before threads
bufsize = la*ha*pr;
bufsize_th = la*ha*pr / (nthreads);
octet_ptr = (unsigned char *)malloc(bufsize);
octet_th = (unsigned char *)malloc(bufsize_th);
(...)
//threads
for(i; i<nthreads; i++)
  pthread_create(&th_a[i], NULL,threadFunction,(void*)(i));

//after threads
for(i=0; i<nthreads; i++)
  pthread_join(th_a[i], NULL);
(...)

void* threadFunction(void* i)
{
(...)
  memcpy(octet_local, octet_ptr+(nthread*bufsize_th), bufsize_th);
  lf=pa/(nthreads);
  nettoie(lf,la,ha,pr,octet_local);
  memcpy(octet_ptr +(nthread*bufsize_th, octet_local), bufsize_th);
(...)
}
```

Foram registados os tempos de execução da limpeza de imagem recorrendo ao OpenMP e às Pthreads e também o tempo de execução sequencial sobre uma imagem de dimensões 200*200*50. O programa correu sobre Linux com versão de Kernel 2.6.22 e o compilador gcc versão 4.2 que suporta o OpenMP. O hardware usado foi um processador dualcore, Intel Core2Duo T5500 com 1.6GHz de velocidade e 2MBs de cache. Foram usadas duas threads (como se pode observar na lista de processos retirada a meio da execução, e que se encontra abaixo), com o Objectivo de distribuir um processo leve por processador. A média de resultados obtidos foi a seguinte:

Media de tempos de execução:

OpenMP: 2 minutos 33 segundos

Pthreads: 2 minutos 35 segundos

Sequencial: 3 minutos 59 segundos

Lista de processos e threads em execução:

PID	TTY	STAT	TIME	COMMAND
				...
5703	pts/1	-	02:17	./limpar imagem.bisegmented imagem.clean
-	-	Rl+	01:06	-
-	-	Rl+	01:10	-
				...

Foram também realizados testes de execução da limpeza num servidor Sun Fire

X4600 M2 x64 com 8 processadores AMD Opteron Model 8220 (2.8GHz-dual-core) memória de 16x2GB DDR2-667 e sistema operativo SunOS fire 5.10 Generic_127128-11 i86pc i386 numa imagem tomográfica de dimensões 1024*1024*476. Foram obtidos os tempos de execução utilizando 16 threads e uma thread para obter uma comparação entre uma execução paralela com vários processadores e uma execução simplesmente sequencial. Os resultados obtidos foram os seguintes:

16 threads - OpenMP: 1 hora 19 minutos 20 segundos

Execução sequencial: 17 horas 35 minutos 21 segundos

Speedup ≈ 13.09

Como podemos observar com a paralelização, tanto recorrendo ao OpenMP, como às POSIX Threads, conseguimos obter resultados significativamente melhores comparados com a execução sequencial. Notam-se as vantagens do processamento paralelo em operações pesadas com apenas uma configuração de processador duplo simples e confirmam-se totalmente as vantagens com uma diferença de tempo de execução muito significativa recorrendo a uma arquitectura com 16 cores. Entre as

duas ferramentas de paralelização por memória partilhada, verificamos que as duas executam em tempos semelhantes. A distinção entre as duas APIs é constatada na estruturação da paralelização que se pode observar nos dois métodos. O OpenMP é uma API de alto nível e claramente mais simples e vantajosa sem necessitar de grande reestruturação do código fonte. Permite também percepcionar a forma como os dados são divididos entre processadores.

4.2.2 Granulometria

Esta função é responsável por detectar partículas nas imagens tomográficas, identificando o centro de massa, as dimensões da partícula e definindo uma caixa que delimita a partícula. Cada vez que a função é executada, cria um ficheiro por cada sub-processo onde são guardados os resultados. A mesma função calcula também o volume e área das partículas. O utilizador pode definir qual o tamanho mínimo para uma partícula ser considerado um blob. Esta operação é muito pesada e como tal necessitou ser paralelizada também.

A operação de granulometria tem tempo de execução é o mais extenso na aplicação. Como referido o objectivo desta função é identificar os compósitos presentes na amostra. O resultado final constituirá de ficheiros de texto. Em cada partícula detectada serão calculadas as coordenadas do centro de massa, um paralelepípedo de delimitação (*bounding box*) que a delimita no espaço tridimensional. É também calculado o volume e superfície de cada partícula. No final será produzido um ficheiro por cada nó. A informação sobre os compósitos ficará então dividida por cada ficheiro.

Esta paralelização foi mais complexa do que a já realizada na limpeza pelo facto da divisão de blocos poder seccionar compósitos que poderão constituir o fundamento da análise tomográfica. Quando um blob for seccionado na divisão de blocos, o programa irá processa-lo como 2 blobs distintos mais pequenos. Este problema verificou-se na tese de Paulo Quaresma com paralelização com memoria distribuída através do MPI. Para solucionar o problema necessitaria-se-ia de detectar os reforços divididos através da sua bounding box que seria o ponto de partida para ligar a

informação dividida entre os ficheiros processados. No final seriam colocados todos os compósitos completos num só ficheiro como mostra a figura 15.

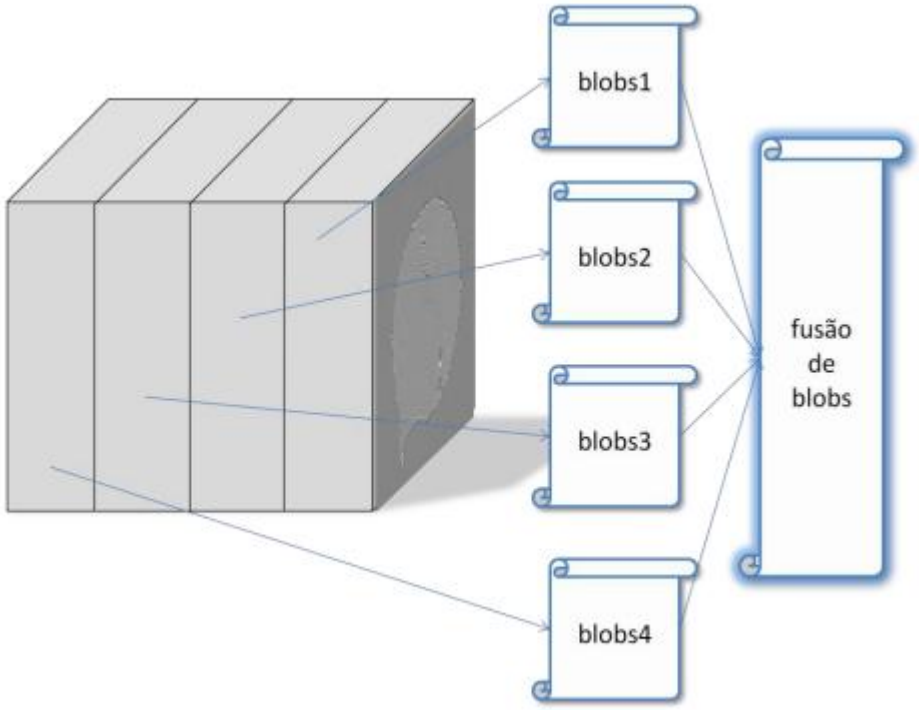

Figura 15: Distribuição de blobs detectados, por ficheiros independentes e posterior agrupamento e fusão de blobs divididos no processamento.

No entanto com memória partilhada através do OpenMP, verificou-se que quando uma *thread* processa uma partícula dividida entre dois blocos, esta vai obter informação sobre o resto da partícula ao bloco vizinho, caso este blob ainda não tenha sido processado anteriormente, simplificando muito o processo. Esta abordagem no entanto pode causar distúrbios de sincronização no caso em que duas *threads* estão a aceder aos mesmos dados num bloco (uma no respectivo bloco e outra no bloco vizinho Para resolver esta situação, foram usados pragmas para sincronização em zonas de código críticas. Isto obriga a que nenhuma thread se sobreponha em cima de outra e quando uma partícula é atribuída a um processo leve, esta já não é atribuída a mais nenhum. Esta solução contudo pode ter custos de eficiência na paralelização pois pode causar esperas entre threads, contudo, pelos testes realizados não foram detectados piorias significativas de tempo de

processamento como se pode ver mais abaixo. De seguida apresentam-se os excertos de código mais relevante na paralelização da granulometria:

```
int main(int argc, char **argv){
(…)
#pragma omp parallel
lista_blobs(lf,la,ha,pr,octet_ptr);
(...)
}

int    lista_blobs(int    kf,int    la,int    ha,int    pr,unsigned
char
*octet_ptr){
(…)
#pragma omp for
    /* Ciclo de percurso da imagem */
 for (i=0;i<kf;i++) {
  threadNum = omp_get_thread_num();
  sprintf(file,"%d.txt",threadNum);
  printf("file: %s\n", file);
  for (j=0;j<ha;j++) {
   for (k=0;k<pr;k++) {
/* O voxel corrente e' preto ? */
     if (octet(octet_ptr,i,j,k,ha,pr) != NOIR) continue;
/* Se for, vai invadindo */
     motor_perco_lb(blobfile,i,j,k,kf,la,ha,pr,pma, pmi,octet_ptr);
/* Algumas estatisticas sobre o blob achado */
     stat_blob(pma,pmi,la,ha,pr,octet_ptr);
/* Repercolacao dentro do blob sem a pele */
     for (ii=pmi->x;ii<=pma->x;ii++)
       for (jj=pmi->y;jj<=pma->y;jj++)
        for (kk=pmi->z;kk<=pma->z;kk++)
motor_reperco(blobfile,ii,jj,kk,kf,la,ha,pr,pma,pmi,octet_pt
r);
        /* Devolver cores de sub-blob */
      pintra_josblob(pma,pmi,ha,pr,octet_ptr);
      pintra_cur((unsigned char) CUR+1, pma,pmi,ha,pr,octet_ptr);
     }
   }
(…)
```

```
}

void    motor_perco_lb(FILE*    blobfile,int    i,int    j,int    k,int
kf,int
la,int ha,int pr,punt *pmax,punt *pmin,unsigned char* octet_ptr){
(…)
#pragma omp critical
        while(nfront>=0) {
       (…)
       /*processamento do blob*/
       (…)
       }
(…)
}
```

Os testes realizados na operação de granulometria foram realizados nas mesmas máquinas dos testes sobre a limpeza (processador com 2 cores e 8 processadores dual core) cujas especificações podem ser revistas acima. A amostra usada para o processador dual core tem dimensões 200*200*110 e a amostra usada no Sunfire de 16 cores tem as dimensões 475*475*43. Os resultados obtidos foram os seguintes:

Processador dual core com amostra 200*200*110:
2 threads - OpenMP: 66 minutos 24 segundos
Execução sequencial: 95 minutos 47 segundos
Speedup ≈ 1.44

Arquitectura de 16 cores com amostra 475*475*43:
16 threads - OpenMP: 4 horas 55 minutos 30 segundos
Execução sequencial: 21 horas 14 minutos 50 segundos
Speedup ≈ 4.32

4.3 Exemplo de aplicação no estudo de uma espuma sintáctica

Para demonstração do potencial do Tritom integrado com o OpenDX, se apresenta, de seguida, resultados aplicados a uma espuma metálica sintáctica funcionalmente graduada que foi apresentada em [38].

As espumas sintácticas são um tipo de *compósitos de matriz metálica* (CMM), em que existem micro esferas cerâmicas ocas (micro balões), assumindo o papel de reforços de compósitos. O conceito de gradiente funcional pode ser estendido a este tipo de compósito, resultando numa espuma sintáctica com gradiente funcional de propriedades (SFGMMC).

Um problema encontrado nestas espumas sintácticas é o facto de alguns destes reforços quebrarem devido à sua fragilidade. Contudo, não é possível averiguar através de imagens bidimensionais, se a causa destes fenómenos são micro balões previamente danificados, ou se por outro lado, se trata de uma manifestação de permeabilidade em esferas intactas. Espera-se que através da micro-tomografia de raios-x, se descubra a resposta a esta questão.

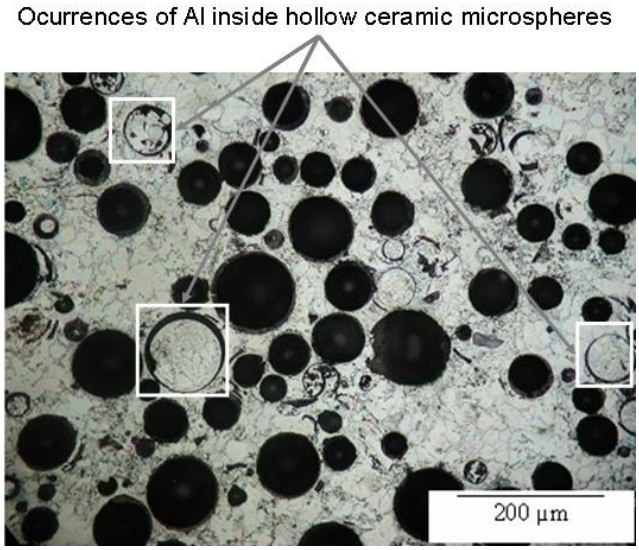

Figura 16: Micrografia óptica de uma FGSCMM onde se verificam esferas quebradas e inundadas. Pela imagem 2-D não é possível verificar se a razão para este

facto são as esferas quebradas ou a permeabilidade das mesmas. Figura retirada de [38].

Para o propósito deste exemplo, foi produzido um compósito SFGMMC em forma de anel por centrifugação. A matriz da espuma consistiu numa liga comercial Al-7Si-0.3Mg reforçada selectivamente com micro balões SiO_2-Al_2O_3 (tamanho médio:50 µm; conteúdo médio dos micro balões: 5 vol. %).

As imagens tomográficas usadas neste trabalho foram obtidas através de um feixe de energia de 20 keV. Para garantir o modo de detecção dos limites, a amostra foi colocada a 100mm do detector, uma câmera FRELON 1024x1024 CCD. O tamanho de pixeis resultante foi 1.3613 µm horizontalmente e 1.3699 µm verticalmente.

Apesar de a vários volumes de interesse serem obtidos de acordo com cada amostra, a atenção deste exemplo foca-se numa representação específica localizada em 11mm a partir da superfície externa do cilindro numa amostra originalmente a 4mm da superfície de topo (figura 17). Este conjunto de dados particular será designado como região de interesse.

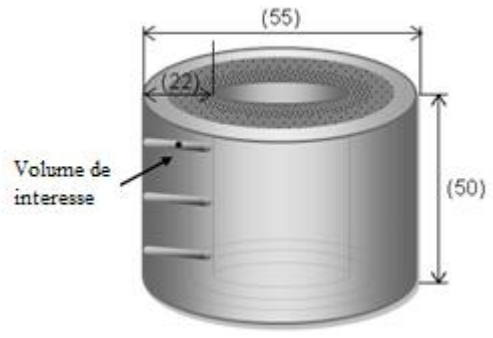

Figura 17: Esquema de um SFGMMC. Retirado de [38].

A figura 18 mostra um exemplo de uma aplicação que integra diversos módulos para a construção de um programa visual de processamento de dados. Este programa efectua uma segmentação na imagem tomográfica de acordo com diferentes parâmetros definidos e modificados interactivamente pelo utilizador

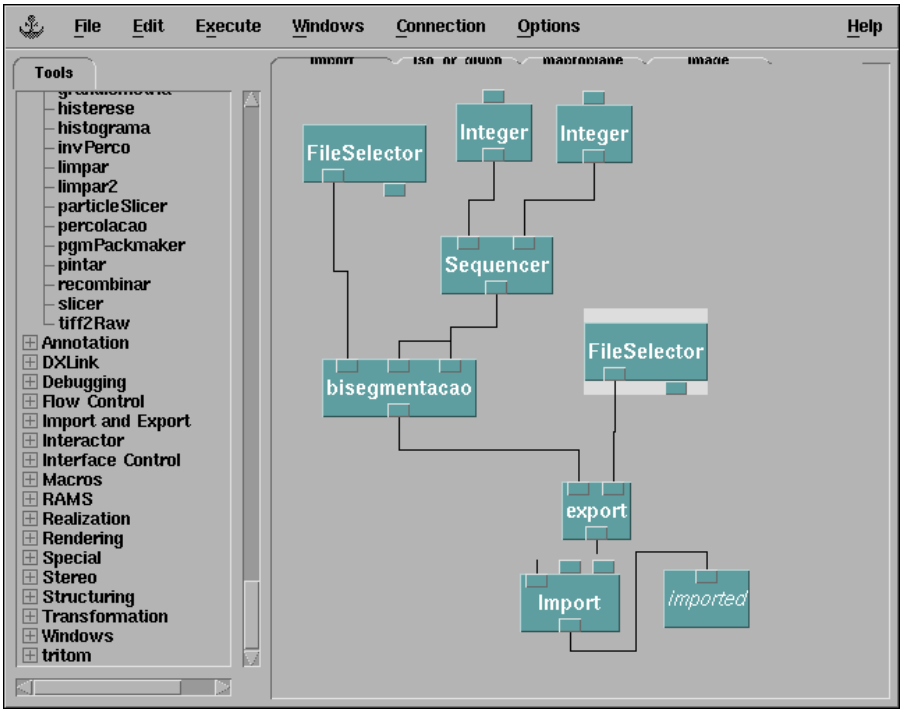

Figura 18: Construção de um programa visual com módulos no OpenDX. Retirado de [38].

Como pode ser visto na figura, os módulos têm parametros entrada e saída que são definidos por um fluxo de dados de acordo com as ligações entre eles. A saída de um módulo é a entrada de outro(s) permitindo uma grande flexibilidade na construção de programas e muitas diferentes formas de processar os dados.

O módulo *bisegmentação* é responsável pelo processamento de segmentação da imagem, recebe como entrada o ficheiro tomográfico e os valores da segmentação necessários para definir as fronteiras na separação da cor. Estes valores passam por um modulo de nome *Sequencer* nativo ao OpenDX, que permite variar estes valores sequencialmente através de uma janela de controlo. Com a alteração interactiva destes valores é possível visualizar os diferentes resultados em tempo real conforme vão sendo processados.

A figura 19 mostra os resultados da operação de segmentação aplicada à região de interesse. As imagens mostram os diferentes resultados de acordo com a variação de volores com o *Sequencer*. Como se pode observar, os resultados apresentam-se sobre a forma de visualização tridimensional.

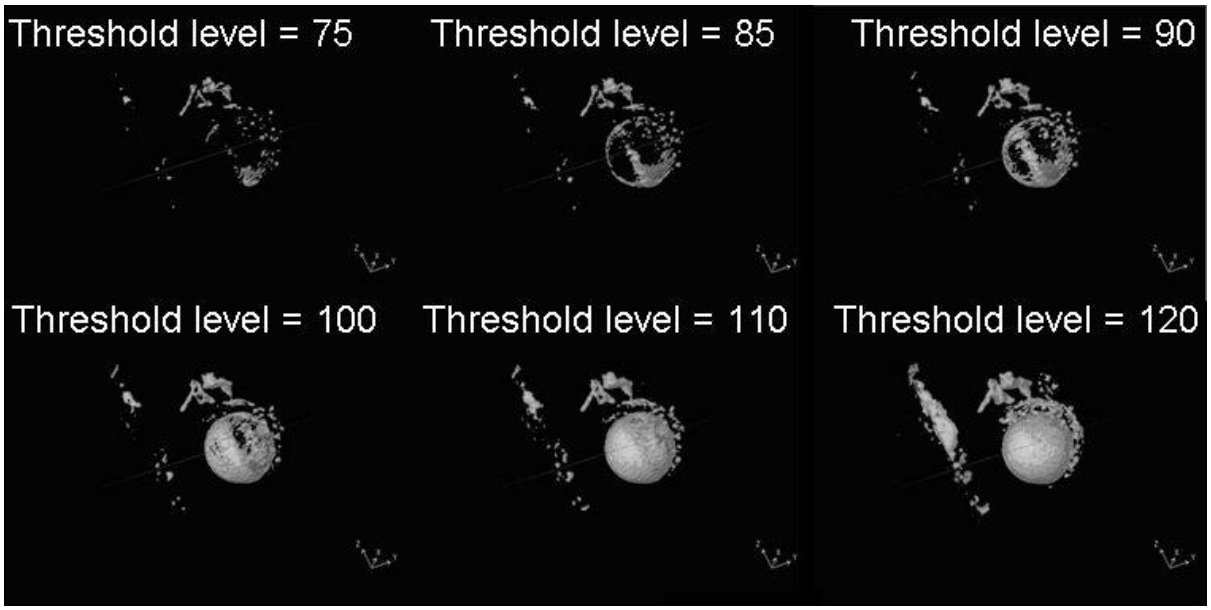

Figura 19: Resultados obtidos pela segmentação na região de interesse. As diferentes imagens correspondem a diferentes valores de segmentação na escala de cinzentos.

Pode-se concluir que existe um valor de segmentação óptimo visto que a reconstrução do micro balão caminha para torna-lo completo, um crescente ruído é computado como pseudo-objectos. Apesar deste efeito poder ser reduzido com um processamento específico para remoção de ruído, é imperativo realizar uma escolha adequada dos valores de segmentação, especialmente devido à espessura de parede dos micro balões que se situa normalmente na ordem dos 3µm.

4.4 Conclusão

Por parte da paralelização pode concluir-se que os efeitos de aceleração de processamento se fazem sentir fortemente com uma grande diminuição no tempo de execução das duas operações demoradas. Com o aumento de cores no processamento, a execução torna-se consideravelmente mais rápida como se pode analisar pela comparação de tempos obtidos na execução das operações demoradas nos diferentes sistemas. Uma vez que o aumento do número de cores e processadores nas arquitecturas de computadores comuns é uma realidade, cada vez mais o Tritom vai tirar partido destas arquitecturas nos computadores novos.

A visualização tridimensional de resultados trouxe vantagens na análise de imagens tomográficas impossíveis de obter de outra forma. Agora é possível visualizar as partículas e algumas falhas nestas, o que era díficil nas imagens bidimensionais.

Com a integração do Tritom no OpenDX sente-se uma melhoria muito significativa na funcionalidade deste programa. Criar programas visuais com os processamentos desejados é agora muito fácil e intuitivo, bastando alguns cliques para seleccionar módulos e interligá-los entre si como se pode ver no exemplo. Com a visualização de resultados intermédios, tornou-se muito mais rápido e simples ajustar os parâmetros de processamento a cada conjunto de dados, visualizando em tempo real os efeitos de cada valor introduzido nos parâmetros, existindo ainda a possibilidade de alterá-los sequencialmente.

Pode então concluir-se por fim que com este trabalho a vida dos investigadores da área de Engenharia de Materiais ficou bastante facilitada, uma vez que o uso do programa Tritom se tornou mais rápido e muito mais simples.

5 Conclusão

5.1 Contribuições atingidas

Com o trabalho realizado nesta tese, foi possível criar um sistema para análise de imagens tomográficas rápido e bastante acessível para os investigadores da área de Materiais.

Através de alguma diversidade de experiências de paralelização por memória partilhada direccionada para os processamentos mais demorados, obtiveram-se melhorias muito significativas, relativas ao tempo de execução. Efectivamente foi conseguida maior rapidez de execução, garantiu-se melhor performance, condições que ofereceram e ampliaram as potencialidades de trabalho aos utilizadores desta aplicação, os investigadores da área de Engenharia de Materiais, favorecendo-lhes seguramente eficácia no seu trabalho. A melhoria de tempos de execução foi verificada tanto em grandes computadores com vários processadores como numa arquitectura bastante simples dual core. Concluímos então que a paralelização por memória partilhada, particularmente através do OpenMP, apresenta consideráveis vantagens em processamentos demorados.

Constatando que nos últimos tempos se tem assistido a um movimento crescente de aumento de processadores e *cores* em computadores de uso comum, as vantagens propostas nesta tese, crescerão em proporção desse aumento tirando sempre o máximo proveito das arquitecturas multi-processador.

Tendo integrado o Tritom no OpenDX, a utilização desta aplicação tornou-se simples e fácil de usar. Os utilizadores podem agora criar os seus próprios programas visuais com processamento que desejam ver realizado de forma gráfica e intuitiva e guardar esses programas para futura referência sempre que necessitarem de voltar a usá-los. A criação destes programas passou a ser realizada através de módulos, sob a forma de rectângulos que os utilizadores podem inserir no programa e interliga-los entre si para formar uma cadeia de processamento. Podem também visualizar resultados em passos intermédios do programa e interactivamente modificar parâmetros de algumas operações para ajustar a processamento à medida do objecto de estudo para obtenção de melhores resultados. Os resultados, tanto finais como intermédios, podem agora ser visualizados de forma tridimensional para analisar alguns fenómenos, antes imperceptíveis com imagens bidimensionais. Para tal, basta inserirem os módulos respectivos que permitem o ajuste da imagem conforme o tipo de visualização que se deseje.

Estão também agora disponíveis no Tritom novas operações correspondendo à solicitação dos investigadores de Ciências dos Materiais, operações que efectivamente se apresentaram indispensáveis para complementar necessidades no processamento tomográfico tais como limpeza de fundo da imagem, conversão de ficheiros tomográficos para o Tritom e recorte da imagem.

Conclui-se afirmando a importância das contribuições atingidas. Os investigadores da área de materiais podem contar agora com a uma ferramenta rápida, eficaz, interactiva, com visualização de dados tridimensionais e de fácil uso para o processamento de dados tomográficos.

Importante também referir que o autor desta tese é também co-autor do artigo [38] submetido para publicação numa revista científica. O sistema descrito nesta tese é uma componente fundamental do trabalho descrito no referido artigo, pois descrevem

a análise de uma amostra tomográfica de uma espuma sintáctica, e cujos compósitos de cerâmica introduzidos necessitam de processamento tomográfico e uma visualização tridimensional para avaliar distúrbios nestes compósitos.

5.2 Trabalho futuro

Em relação aotrabalho futuro, o processamento e a visualização de dados tomográficos poderá vir a ser desenvolvido em vários aspectos.

Atendendo a que os investigadores da área de Materiais, em particular na área de materiais compósitos, introduzem continuamente novas técnicas e novos materiais e esta actividade conduz naturalmente a novas necessidades de desenvolvimento de módulos de processamento. O sistema desenvolvido em torno do OpenDX permite integrar facilmente estes novos tratamentos, sob a forma de módulos.

Poderão vir a ser acrescentadas mais operações ao Tritom para processamento tomográfico, nomeadamente um módulo para lidar com casos especiais de partículas ocas que estejam inundadas devido a infiltração na sua porosidade ou simplesmente quebradas. Estes novos processamentos poderão conduzir a tempos de execução muito longos, introduzindo assim a necessidade de os paralelizar.

4. Bibliografia

[1] Paulo Jorge Lago da Silva Quaresma, "*Processamento paralelo aplicado a um problema de Engenharia de Materiais*", MSc Thesis, Lisboa, Abril de 2007

[2] Alexandre Velhinho, "*Fundição centrífuga de compósitos alumínio/sic com gradiente funcional de propriedades: Processamento e caracterização*", Ph.D. thesis, Departamento de Ciências dos Materiais - Universidade Nova de Lisboa, 2003.

[3] W.A. Kalender, "*Computed Tomography*", ed Publicis MCD Verlag, Munich, 2000.

[4] M. Flynn. "Very *high speed computing systems*", Proceedings of the IEEE, 1901–1909, Dec 1966

[5] G. M. Amdahl, "*Validity of the single processor approach to achieving large scale computing capabilities*", In AFIPS Conference Proceedings, vol. 30, 483–485, AFIPS Press, 1967

[6] I. Foster, "*Designing and Building Parallel Programs*", Addison Wesley, 1995

[7] Hesham El-Rewini and Mostafa Abd-El-Barr, "*Advanced Computer Architecture and Parallel Processing*", Wiley, USA, 2005

[8] Micheal J. Quinn, "*Parallel Programming in C with MPI and OpenMP*" 1st edition, Mc Graw Hill, 2004

[9] Barbara Chapman, Gabrielle Jost and Ruud Van Der Pas, "*Using OpenMP - Portable Shared Memory Parallel Programming*", Scientific and Engineering Computation, The MIT Press, 2008

[10] Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon and Andy White, "*Sourcebook of Parallel Computing*", Morgan Kaufmann Publishers, 2003

[11] David Culler, J.P. Singh, Anoop Gupta, "*Parallel Computer Architecture: A Hardware/Software Approach*", Morgan Kaufmann Publishers, 1998

[12] William Gropp, Ewing Lusk and Thomas Sterling, "*Beowulf Cluster Computing with Linux*", 2nd Edition, MIT Press, 2003

[13] Cliford A. Shaffer, Layne T. Watson, and Dennis G. Kafura, "*Component Frameworks for Problem Solving Environments in Computational Science*", in Proc. First Symposium on Reusable Architectures and Components for Developing Distributed Information Systems, 1999 pp.653-656

[14] Ian Taylor, Matthew Shields, Ian Wang and Omer Rana, "*Triana Applications within Grid Computing and Peer to Peer Environments*", Journal of Grid Computing 2003, vol. 1, no. 2, pp. 199–217, 2003.

[15] Ian J. Taylor, Matthew S. Shields,Ian Wang and Andrew Harrison, "*Visual Grid Workflow in Triana*", Journal of Grid Computing 2005, vol 3, no. 3-4, pp. 153-169, 2005

[16] G. Fox, D. Gannon, and M. Thomas, "*A Summary of Grid Computing Environments*", Concurrency and Computation: Practice and Experience (Special Issue), 2003.

[17] F. Garcia and J. Fernandez, "*POSIX thread libraries*", Linux Journal, (70), 2000.

[18] G. Vignoles, "*Image segmentation for phase-contrast hard X-ray CMT of C/C composites*", Carbon 39, 167-173, 2001

[19] S. Ferreira, L. Rocha, G. Vignoles, P.Cloetens, A. Velhinho and B. Fernandes, "*Application of x-ray microtomography to the microstructural characterization of al-based functionally graded materials"*, 2001.

[20] S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman, "*Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience*", Proceedings of the 9th Heterogeneous Computing Workshop, May 2000.

[21] Shava Smallen e Henri Casanova e Francine Berman, "*Applying scheduling and tuning to on-line parallel tomography*", 2001.

[22] D. A. Reimann, V. M. Chaudhary, J. Flynn, and I.K. Sethi, "*Parallel Implementation of Cone-Beam Tomography*", International Conference on Parallel Processing, II, pp. 170-173, 1996.

[23] C. Laurent, F. Peyrin, J.M. Chassery, and M. Amiel, "*Parallel image reconstruction on MIMD computers for three-dimensional conebeam tomography*", Parallel Computing 24 , 1461–1479., 1998

[24] S. Tomov , R. Bennett , M. McGuigan , A. Peskin , G. Smith and J. Spiletic, "Application of interactive parallel visualization for commodity-based clusters using visualization APIs", Computers & Graphics, 28, 273, 2004.

[25] Gregor Laszewsky e Joseph A. Insley e Ian Foster e John Bresnaham, "Real-time analysis, visualization, and steering of microtomography experiments at photon sources", *Proceedings of the Ninth {SIAM} Conference on Parallel Processing for Scientific Computing*, Philadelphia- Pennsylvania/USA, 1999.

[26] M. Pereira, "*Algoritmo paralelo para reconstrução tridimensional de imagens tomográficas de amostras agrícolas em arquitetura dsp com técnicas wavelets"*, Tese de Mestrado, Universidade Federal de São Carlos-São Paulo/Brasil, 2001.

[27] Pedro D. Medeiros, "*Introdução ao processamento paralelo*", FCT-UNL, 1998.

[28] Blaise Barney, "POSIX Threads Programming", https://computing.llnl.gov/tutorials/pthreads/, Livermore Computing, visitado na actualização de 04/02/2008.

[29] OpenMP homepage, http://www.openmp.org/, visitado na actualização de 05/11/2007.

[30] Behrooz Parhami, "Introduction to parallel processing", Springer, 1999.

[31] Chromium homepage website, http://chromium.sourceforge.net/, visitado em 10/01/08.

[32] Open Inventor homepage, http://oss.sgi.com/projects/inventor/, visitado em 10/01/08.

[33] VTK homepage, http://www.vtk.org/, visitado em 10/01/08.

[34] Volumizer homepage, http://www.sgi.com/products/software/volumizer/, visitado em 10/01/08

[35] Bruce Lucas, Gregory D. Abram, Nancy S. Collins, David A. Epstein, Donna L. Gresh, Kevin P. Mc Auliffe, "An Architecture for a Scientific Visualization System", IBM T.J. Watson Research Center, 1992

[36] OpenDX homepage, http://opendx.org/, visitado em 20/05/08

[37] David Thompson, Jeff Braun, Ray Ford, "OpenDX Paths to Visualization", Visualizationand Imagery Solutions, second edition, 2004

[38] T. Cadavez, S.C. Ferreira, P. Medeiros, P.J. Quaresma, L.A. Rocha, A. Velhinho, G. Vignoles, "A Graphical Tool for the Tomographic Characterization of Microstructural Features on Metal Matrix Composites", artigo aceite para publicação na revista "International Journal of Ttomography and Statistics", Special Issue on Image Processing, 2008

[39] AVS homepage, http://www.avs.com/, vistado em 20/07/08

[40] Brian Sheeman, Stephen D. Fuller, Michael E. Pique, Mark Yeager, "AVS Sofware for Visualization in Molecular Microscopy", Journal of Structural Biology 116, 99-106, 1996

[41] Mercury Avizo homepage, http://www.tgs.com/products/avizo.asp, visitado em 20/05/08

[42] Matej Daniel, Aleš Iglič, Veronika Kralj-Iglič, Svatava Konvičková, "Computer system for definition of the quantitative geometry of musculature from CT images", Taylor & Francis, 2005

[43] Paul Grosu, Jeffrey P. Townsend, Daniel L. Hartl, Duccio Cavalieri, "Pathway Processor: A Tool for Integrating Whole-Genome Expression Results into Metabolic Networks", Cold Spring Harbor Laboratory Press, 2008